Programming for Performance – Part 2

XMT Performance-Tuning Tools

The C/C++ Compiler: Initial Comments

- **The XMT programmer needs to think about *algorithms* and the *compiler.***
- **Running XMT programs is, practically speaking, dependent on the XMT C/C++ compiler.**
- **Programming the XMT for performance is a "negotiation" with the compiler.**

The XMT C/C++ Compiler: Optimizing Loop-Level Parallelism

– with thanks to Mike Ringenburg

Outline

Introduction to loop parallelism

- Conditions necessary for parallelism
- Single processor, multiprocessor, and loop future parallelism

Pragmas to assist parallelization

- The noalias pragma and the restrict type qualifier
- The no dependence pragma
- The assert parallel pragma

Compiler transformations to augment parallelism

- Scalar expansion
- Reductions/Recurrences
- Nested parallelism and loop collapse

A parallelization example

When Will the Compiler Parallelize a Loop?

The compiler attempts to parallelize your loops if:

1. **It can figure out how to compute the number of iterations prior to executing the loop**
2. **It can prove that there are no dependences between iterations**
3. **There are no function calls with unknown side effects (e.g., output)**
4. **The loop has a simple structure (e.g., no multiple exits)**

Pragmas are promises made by the user that help the compiler establish that these conditions hold.

Example

This loop parallelizes:

```
void foo(int n) {
      int i;
      int my_array[n];
      for (i = 0; i < n; i++) {
            my_array[i] = i;
      }
      return;
}
```

Example 2

This loop does not:

```
void foo(int *a, int *b) {
     int i;
     for (i = 0; i < 10000; i++) {
          a[i] = b[i];
     }
}
```

a and b may point to overlapping memory

```
foo(x+5000, x);
```

The Three Forms of Parallelism

There are three forms of loop parallelism available on the XMT: single processor, multiprocessor, and loop futures.

- **You can select a preferred mode with a compile flag**
  - `-par` **for multiprocessor (this is the default),** `-par1` **for single processor, and** `-parfuture` **for loop futures**
- **In multiprocessor mode, the compiler will sometimes choose to use single processor if it judges that the amount of work and iterations are too small to justify the overhead of multiprocessor.**
- **You can override the parallelization mode with a pragma**
- **Parallelization mode is determined on a per-region basis**

Single Processor Parallelism

```
#pragma mta loop single processor
for (int i = 0; i < small_size; i++)
  a[i] = b[i];
```

- Use multiple threads on a single processor.
- Very low overheard.
- Good for shorter loops where the time saved by going parallel does not justify the expense of more heavy-weight forms of parallelism.

Multiprocessor Parallelism

```
#pragma mta loop multiprocessor
for (int i = 0; i < big_size; i++)
  a[i] = b[i];
```

- Use multiple threads on multiple processors.
- Higher overhead.
- Allows you to take advantage of all the resources of the machine.

Loop Future Parallelism

```
#pragma mta loop future
for (i = firstNode; i < lastNode; i++) {
  int nbr = Neighbors[i];
  int v = int_fetch_add(&Visited[nbr], 1);
  if (v == 0) BFS(nbr, A);
}
```

- Loop futures are a highly dynamic style of loop parallelism
  - For those familiar with futures, this is *not* just a loop of futures
  - Compiler still manages threads and schedules iterations
- Highest overhead form of loop parallelism
- The only form of parallelism where the number of assigned threads can increase dynamically
- Good for recursive-style loops with highly variable workloads

Outline

Introduction to loop parallelism

- Conditions necessary for parallelism
- Single processor, multiprocessor, and loop future parallelism

Pragmas to assist parallelization

- The noalias pragma and the restrict type qualifier
- The no dependence pragma
- The assert parallel pragma

Compiler transformations to augment parallelism

- Scalar expansion
- Reductions/Recurrences
- Nested parallelism and loop collapse

A parallelization example

Using Pragmas to Help Find Parallelism

The XMT compiler supports a number of pragmas that can be used to give the compiler additional information about loops and the variables referenced inside them. The most commonly used are:

1. `pragma mta assert noalias`
2. `pragma mta assert no dependence`
3. `pragma mta assert parallel`

The compiler treats these pragmas as promises by the user

- **The compiler trusts what you tell it**
- **If you give incorrect information, and the compiler relies on it, your program may not run correctly.**

The noalias Pragma and restrict

```
void foo(int *x, int*y, int*z) {
  #pragma mta noalias *x, *y
  for (int i = 0; i < N; i++) {
    z[i] = x[i] + y[i];
  }
}
```

- Promises that the listed variables are not aliased with any other variables.
- Must appear within the scope and after the declarations of the listed variables.
- Only need to use once per variable (not once per loop).

The noalias Pragma and restrict

```
void foo(int* restrict x, int* restrict
y,
         int* z) {
  for (int i = 0; i < N; i++) {
    z[i] = x[i] + y[i];
  }
}
```

- Promises that the listed variables are not aliased with any other variables.
- Must appear within the scope and after the declarations of the listed variables.
- Only need to use once per variable (not once per loop).
- Can also use `restrict` pointers to get the same affect.

The no dependence Pragma (or nodep)

```
#pragma mta assert noalias *IA
#pragma mta assert no dependence *IA
for (int i = 0; i < N; i++) {
  IA[i][1] = IA[i][INDEX[i]];
}
```

- Promises that any memory location accessed in the loop via any variable on the `no dependence` list is accessed by *exactly* one iteration of the loop
- Appears immediately before a loop
- Variables must be `noalias` or `restrict` pointers

The no dependence Pragma (or nodep)

```
#pragma mta assert no dependence
for (int i = 0; i < N; i++) {
  IA[i][1] = IA[i][INDEX[i]];
}
```

- Promises that any memory location accessed in the loop via any variable on the `no dependence` list is accessed by *exactly* one iteration of the loop
- Appears immediately before a loop
- Variables must be `noalias` or `restrict` pointers
- Can also use with no variable list. This makes the pragma apply to all memory references in the loop (and doesn't require noalias pragmas).

The assert parallel Pragma

```
#pragma mta assert parallel
for (int i = 0; i < N; i++) {
  printf("May appear out of order %d",i);
}
```

- Promises that the iterations of the loop can safely be executed concurrently without any synchronization.
- Does not force the compiler to parallelize the loop, but it is a strong suggestion.
- Should only be used when other techniques to get your loop to parallelize fail. It limits the types of optimizations and transformations the compiler can perform on the loop.
  - You are only asserting that the loop is parallel *as written*.
  - Compiler worries that loop transformations may invalidate that.

Outline

Introduction to loop parallelism

- Conditions necessary for parallelism
- Single processor, multiprocessor, and loop future parallelism

Pragmas to assist parallelization

- The noalias pragma and the restrict type qualifier
- The no dependence pragma
- The assert parallel pragma

Compiler transformations to augment parallelism

- Scalar expansion
- Reductions/Recurrences
- Nested parallelism and loop collapse

A parallelization example

Compiler Transformations for Parallelism

The compiler will attempt to restructure code to find or enhance parallelism:

- **Scalar expansion**
- **Reductions**
- **Loop collapse**

You can view the ways the compiler restructured your code in Canal (text-based) or in the Canal report of Apprentice2 (GUI-based).

Scalar Expansion

This loop can not be parallelized as written because of dependences between the reads and writes of `t` in different iterations (writing `t` in one iteration may overwrite the value of `t` from another iteration before it is used):

```
int t;
for (i = 0; i < n; ++i) {
  t = sqrt(b[i]);
  ...
  a[i] = t + 5;
}
```

Scalar Expansion

This loop can not be parallelized as written because of dependences between the reads and writes of `t` in different iterations (writing `t` in one iteration may overwrite the value of `t` from another iteration before it is used):

```
int t;
for (i = 0; i < n; ++i) {
  t[i] = sqrt(b[i]);
  ...
  a[i] = t[i] + 5;
}
```

The compiler solves this by converting the scalar integer `t` into an array of integers

Scalar Expansion

Viewing this loop in the Canal report of Apprentice2, we see:

```
        | int t;
        | for (i = 0; i < n; ++i) {
5 P:e   |   t = sqrt(b[i + 1]);
        |   ...
5 P     |   a[i] = t + 5;
        | }
```

Performed scalar expansion

Reductions

The compiler attempts to recognize loops that calculate sums, products, minimums, and maximums over an array. E.g.:

```
int min = MAX_VAL;
for (i = 0; i < n; i++) {
  if (x[i] < min)
    min = x[i];
}
```

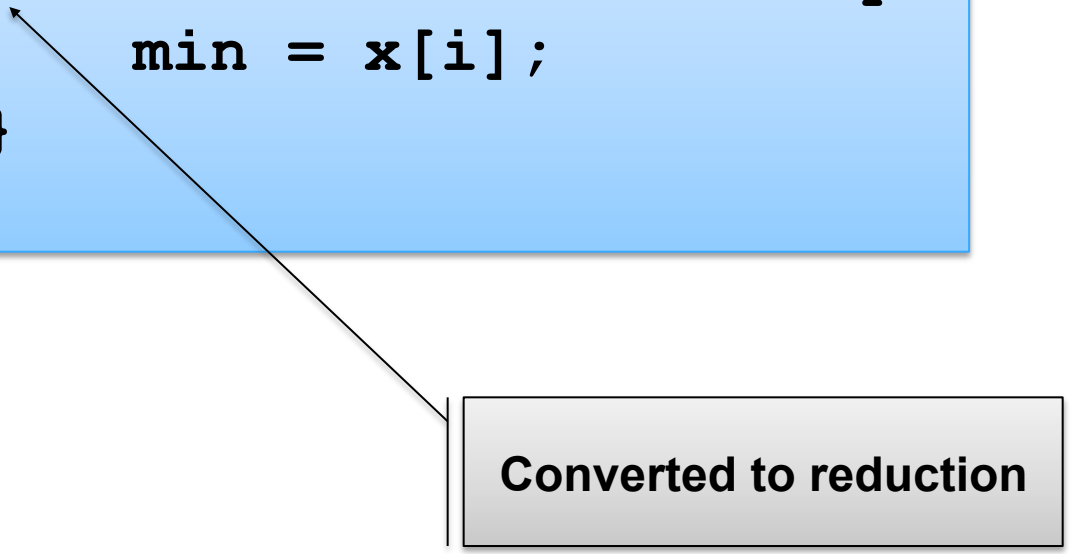
The compiler converts these to reductions

- **Each thread computes the min/max/sum/product over a sub-section of the array.**
- **Threads then combine results to determine the final value.**

Reductions

Viewing this in the Canal report of Apprentice2, we see:

```
        | for (i = 0; i < n; i++) {
3 P:$   |   if (x[i] < min)
** reduction moved out of 1 loop
        |     min = x[i];
        | }
```



Nested Parallelism

```
void foo(int* restrict num_bars, int size_x,
         int* restrict x, int* restrict bar) {
  for (int i = 0; i < size_x; i++)
    for (int j = 0; j < num_bars[i]; j++)
      x[i] += bar[i + j];
}
```

- How do we handle nested parallel loops?
- Option 1: Go parallel for the outer loop, and then again for the inner loop.
  - Inefficient – there is a significant overhead to going parallel. If we nest, then every iteration of the outer loop has to pay that overhead.
  - Limits the effectiveness of the load balancing obtained by some of the scheduling methods.

Loop Collapse

```
void foo(int* restrict num_bars, int size_x,
         int* restrict x, int* restrict bar) {
  for (int i = 0; i < size_x; i++)
    for (int j = 0; j < num_bars[i]; j++)
      x[i] += bar[i + j];
}
```

Option 2: Loop collapse.

- Convert the nested pair of parallel loops to a single parallel loop that simulates the execution of the nested loops.
  - Create a new parallel loop to calculate the total number of iteration of the inner loop (across all iterations of the outer loop).
  - Convert the pair of loops into a single loop where each iteration corresponds to a distinct outer/inner iteration pair.

Often a big performance win.

Collapse Psuedocode

```
// t[i] = total # of inner loop iterations
// in first i iterations of outer loop
t[0] = 0;
for (i = 0; i < size_x; i++)
  t[i + 1] = t[i] + num_bars[i];

for (k = 0; k < t[size_x]; k++) {
  // Set i to index of largest element of t
  // less than k (use binary search)
  i = max_element_less_than(t,  k);
  j = k - t[i];

  x[i] += bar[i + j]; // original loop body
}
```

Loop Collapse in Canal

```
          |  for (int i = 0; i < size_x; i++) {
          |    for (int j = 0; j < num_bars...
4 PP:m$   |      x[i] += bar[i + j];
** reduction moved out of 1 loop
```

‘m’ indicates loop collapse occurred

Outline

Introduction to loop parallelism

- Conditions necessary for parallelism
- Single processor, multiprocessor, and loop future parallelism

Pragmas to assist parallelization

- The noalias pragma and the restrict type qualifier
- The no dependence pragma
- The assert parallel pragma

Compiler transformations to augment parallelism

- Scalar expansion
- Reductions/Recurrences
- Nested parallelism and loop collapse

A parallelization example

An Example

```
bool foo(int *a, int *b, int n,
         int sought, int *old_val) {
  int i;
  for (i = 0; i < n; i++) {
    if (b[i] == sought)
      break;
    a[i] = b[i];
  }
  return (i < n);
}
```

An Example (2)

```
 1 X |    for (i = 0; i < n; i++) {
** loop exit
** multiple exits
     1 X |      if (b[i] == sought)
         |        break;
     1 X |      a[i] = b[i];
         |    }
```

An Example (3)

```
bool foo(int *a, int *b, int n,
         int sought, int *old_val) {
  int i;
  int found_index = n;
  for (i = 0; i < n; i++) {
    if (b[i] == sought)
      if (i < found_index)
        found_index = i;
  }
  for (int i = 0; i < found_index; i++)
    a[i] = b[i];
  return (found_index < n);
}
```

An Example (4)

```
       |    for (i = 0; i < n; i++) {
3 P:$  |      if (b[i] == sought)
** reduction moved out of 1 loop
       |        if (i < found_index)
       |          found_index = i;
       |    }
       |    for (int i = 0; i < found_index; i++)
4 S    |      a[i] = b[i];
```

An Example (5)

```
bool foo(int *a, int *b, int n,
         int sought, int *old_val) {
#pragma mta assert noalias *a
  int i;
  int found_index = n;
  for (i = 0; i < n; i++) {
    if (b[i] == sought) {
      if (i < found_index) {
        found_index = i;
  }
  for (int i = 0; i < found_index; i++)
    a[i] = b[i];
  return (found_index < n);
}
```

An Example (6)

```
      |  #pragma mta assert noalias *a
      |    int i;
      |    int found_index = n;
      |    for (i = 0; i < n; i++) {
3 P:$|      if (b[i] == sought) {
** reduction moved out of 1 loop
      |        if (i < found_index) {
      |          found_index = i;
      |    }
      |    for (int i = 0; i < found_index; i++)
5 P   |      a[i] = b[i];
```

Summary

Loop parallelism is an important technique for obtaining good performance on the XMT.

The compiler will automatically parallelize loop if it can establish that it is safe to do so.

- **Safe means that parallelization will preserve the correct program behavior.**

Pragmas may be used to assist the compiler in proving safety.

The compiler will also attempt to aggressively transform loops to make them safe to parallelize.

Reading the Canal Output

Canal parallel annotations

Code	Description
P	The compiler parallelized the loop automatically.
p	An assertion caused the compiler to parallelize the loop.
D	The compiler parallelized the loop, even though it looks like there is a dependence between iterations.
L	The loop is a linear recurrence or reduction that the compiler parallelized.

Reading the Canal Output

Canal sequential annotations

Code	Description
-	The compiler could not parallelize the loop.
S	The marked statement prevented the compiler from parallelizing the loop.
s	The compiler could have parallelized the loop, but didn't because the loop had too few iterations.
X	The compiler didn't parallelize the loop, because the loop isn't inductive.

Additional annotations

Code	Description
U	The compiler completely unrolled the loop.
e	The compiler expanded the scalar variable in the statement to a vector that it distributed across the loop iterations.
$	The compiler implemented the memory update in this statement as an atomic operation, using full-empty bits.

Exercise 3

```
/mnt/lustre/Workshop/Exercise3
```

Lunch

Using Canal, Traceview and Bprof in Apprentice2

– with thanks to Tracy Hartford

Overview

- Apprentice2
  - GUI application for debugging performance problems
  - Consists of one or more reports based on how your program was compiled and executed
- Canal Report
  - Feedback from the compiler
  - Insight on how latent parallelism was exploited
  - Information on expected resource utilization and scheduling
- Tview Report
  - Hardware counter plots
  - Actual performance of your application
  - Runtime trap information for detecting hotspots
- Bprof Report
  - Profile tables in terms of instructions issued and memory references

Outline

- **Two sample workflows**
  - Provide an overview of all the visual elements of Apprentice2
  - Highlight the interactions between the reports
- **Parallelizing and tuning a radix sort**
- **Removing a hotspot from a BFS-like queue**

Workflow: Radix Sort I

- **Write code**
  - Sequentially sorts on each byte of an eight byte word, starting with the least significant byte and preserving the relative ordering from partial sort to partial sort.
  - Count the elements with the same value at the current byte position:

```
for (i = 0; i < size; ++i) {
  cnt[MTA_BIT_PACK(~mask, src[i])]++;
}
```

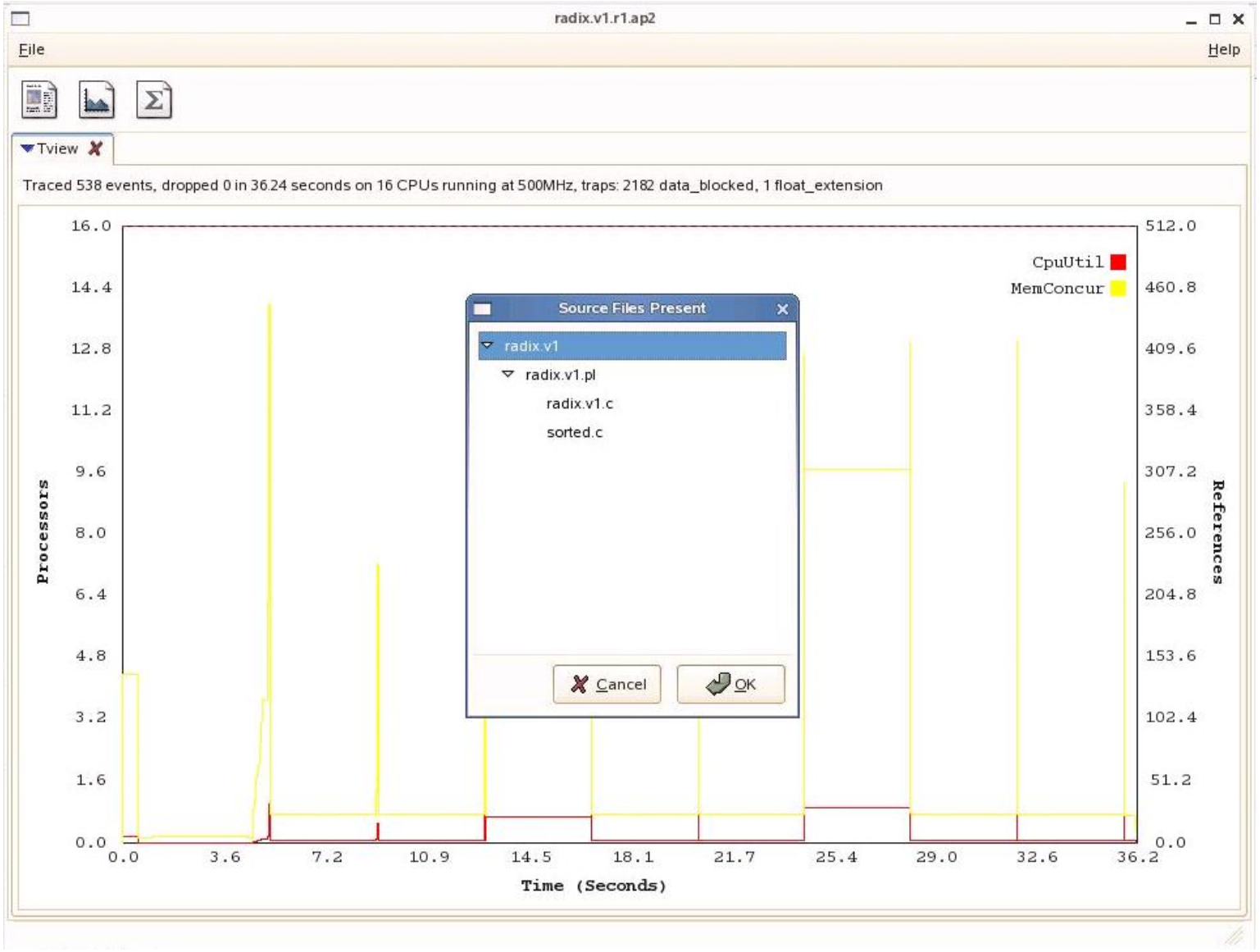
- **Compile**
  - `cc –o radix.v1 sorted.c radix.v1.c`
- **Post-process**
  - `pproc radix.v1`
- **Run Apprentice2**
  - `app2 radix.v1.ap2 [additional .ap2 files]`

Workflow: Radix Sort II

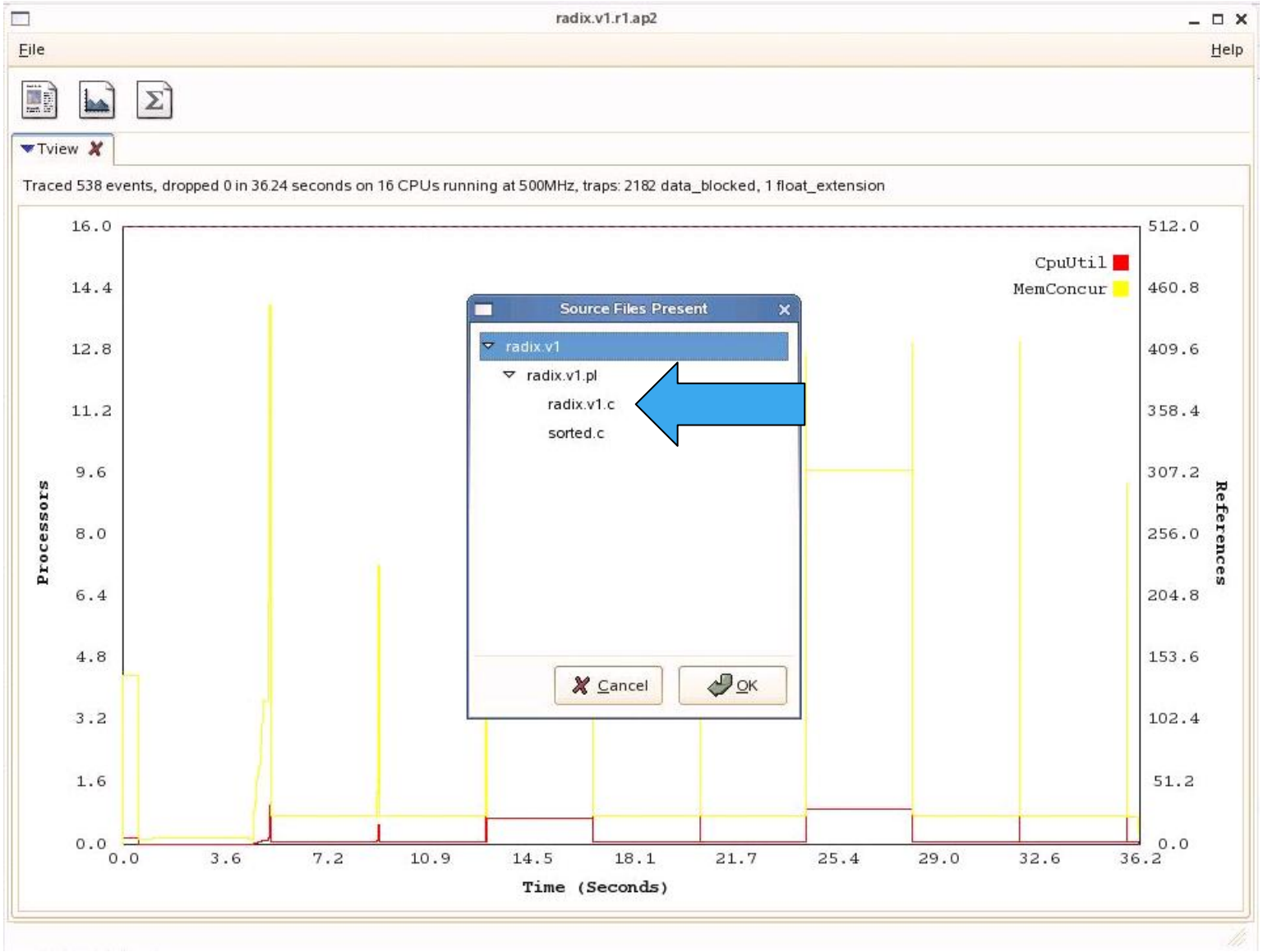
- **Modify code**
  - Add user trace events to further identify areas of interest
  - Move the elements to their new relative positions

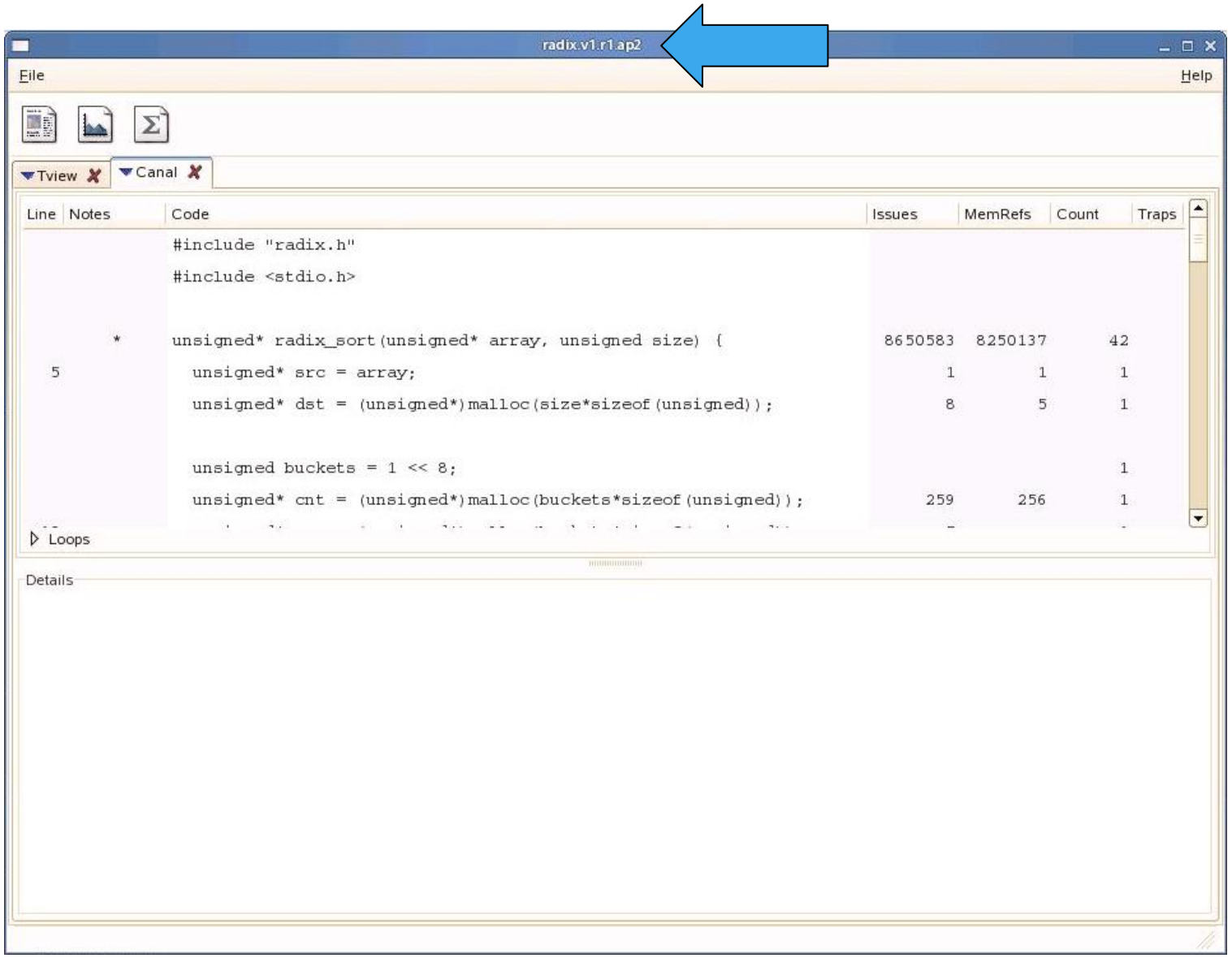
```
#pragma mta trace "shift elements"
    for (i = 0; i < size; ++i) {
      j = pos[MTA_BIT_PACK(~mask, src[i])]++;
      dst[j] = src[i];
    }
```

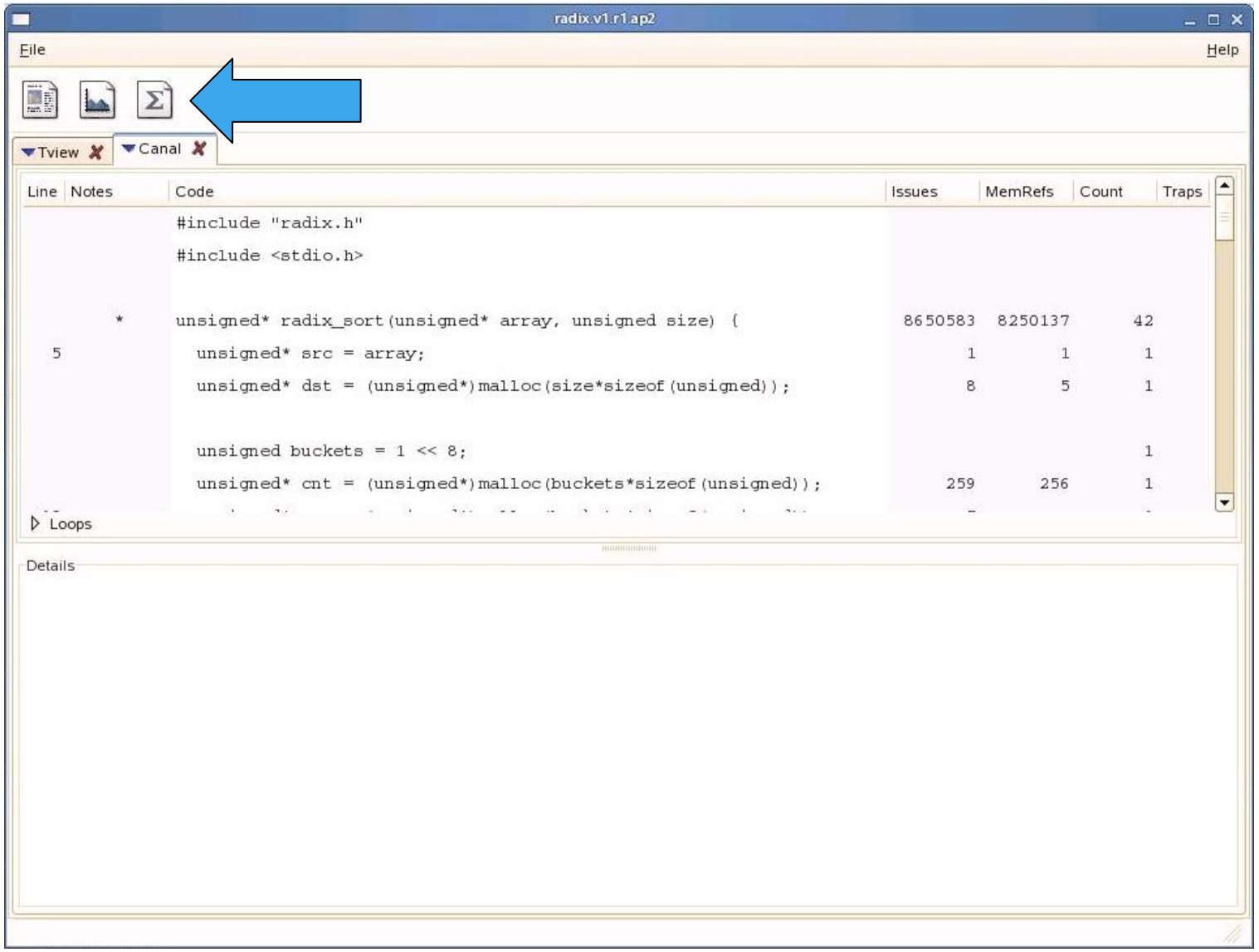
- **Recompile with tracing and profiling enabled**
  - `cc -trace -profile -o radix.v1 sorted.c radix.v1.c`
- **Run the application and post-process**
  - `mtarun -trace -pproc radix.v1`
- **Run Apprentice2**
  - `app2 sorted.v1.ap2`

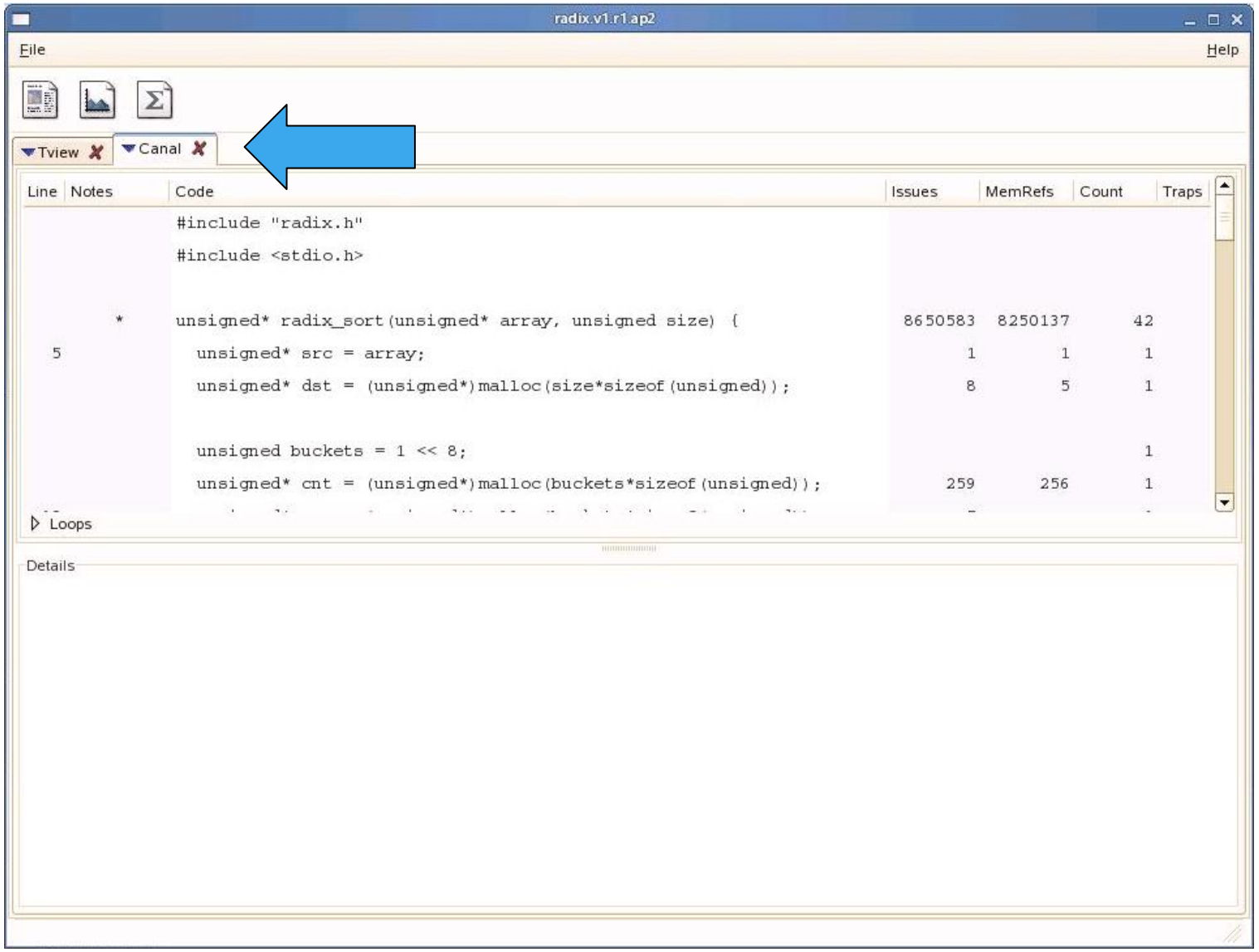

radix.v1.r1.ap2
File
Help
Tview
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
Source Files Present
radix.v1
radix.v1.pl
radix.v1.c
sorted.c
Cancel
OK
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

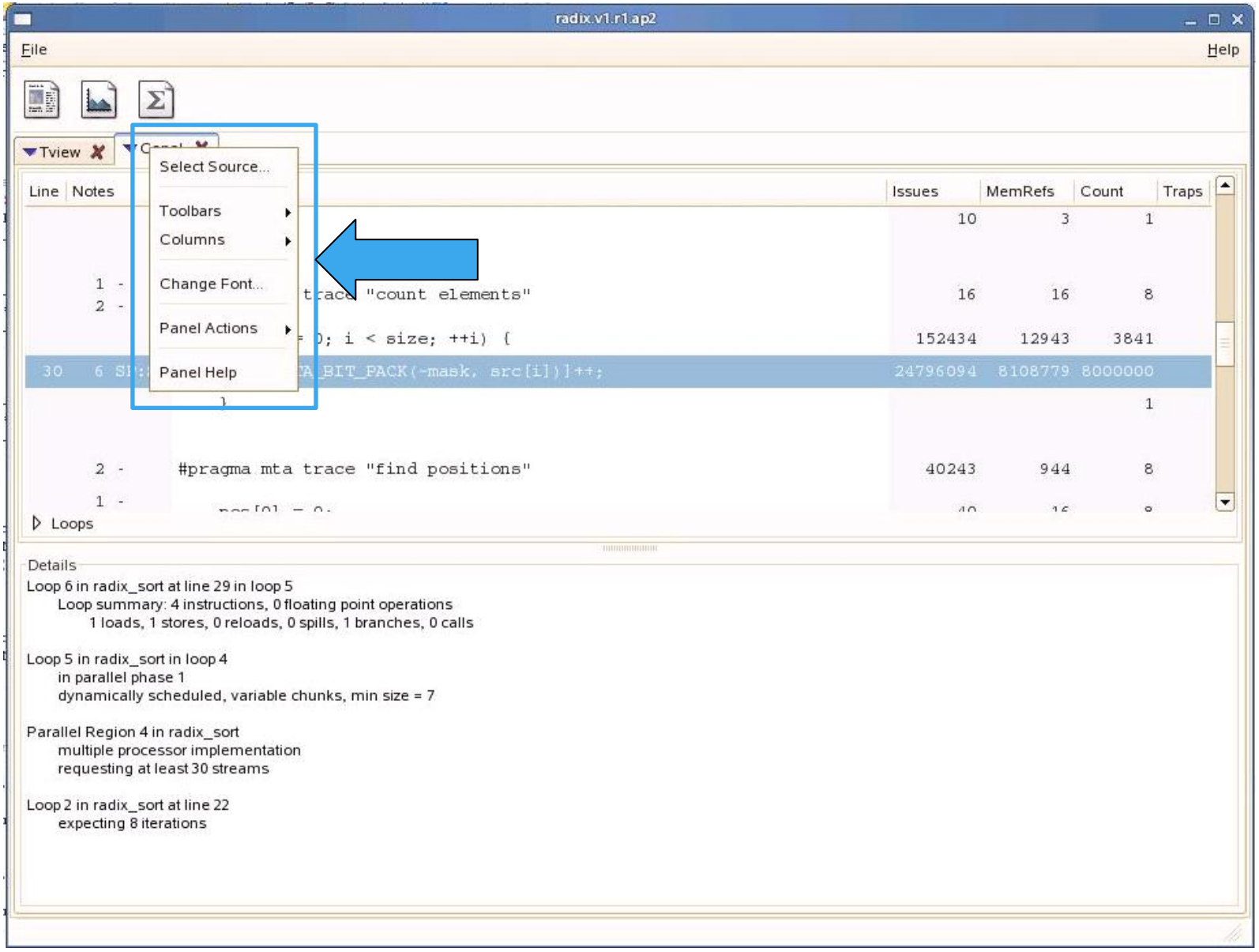

Source Files Present
rmat2file.x
rmat2file.pl
./mtgl/dynamic_array.h (graph.o)
./mtgl/dynamic_array.h (rmat2file.o)
./mtgl/graph_adapter.hpp
./mtgl/graph.h
./mtgl/mtgl_adapter.hpp
./mtgl/mtgl_boost_graph.hpp
./mtgl/rand_fill.hpp
./mtgl/util.h (graph.o)
./mtgl/util.h (rmat2file.o)
./mtgl/xmt_hash_table.hpp
/opt/mta-pe/6.4.0/include/c++/exception (graph.o)
/opt/mta-pe/6.4.0/include/c++/exception (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/iostream (graph.o)
/opt/mta-pe/6.4.0/include/c++/iostream (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/new (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/new (graph.o)
/opt/mta-pe/6.4.0/include/c++/stdexcept (graph.o)
/opt/mta-pe/6.4.0/include/c++/stdexcept (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/typeinfo (graph.o)
/opt/mta-pe/6.4.0/include/c++/typeinfo (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/xiosbase (graph.o)
/opt/mta-pe/6.4.0/include/c++/xiosbase (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/xlocale (graph.o)
/opt/mta-pe/6.4.0/include/c++/xlocale (rmat2file.o)
/opt/mta-pe/6.4.0/include/c++/xstring (graph.o)
/opt/mta-pe/6.4.0/include/c++/xstring (rmat2file.o)
test/graph.cpp
test/rmat2file.cpp
test/static_graph.c
Cancel
OK

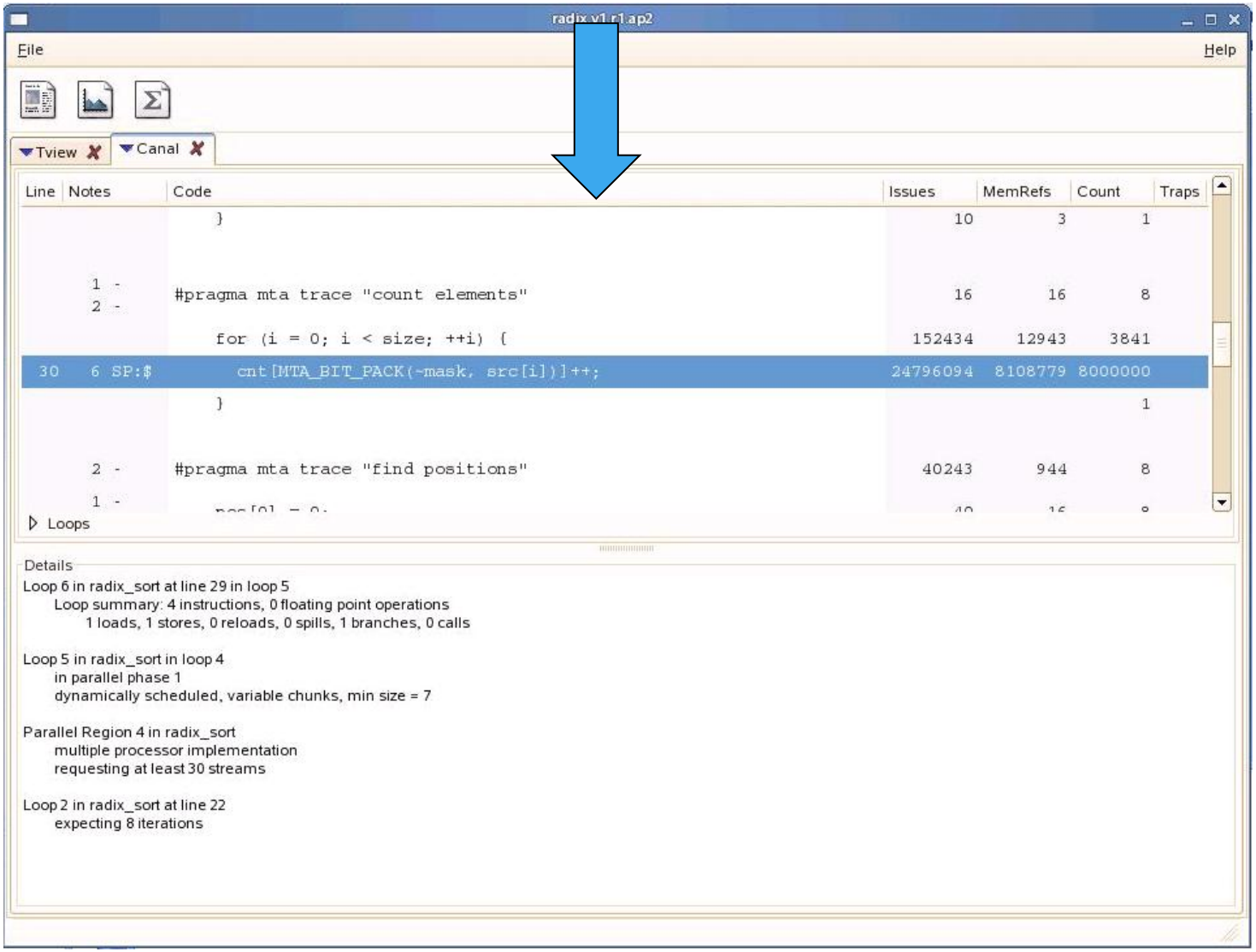

radix.v1.r1.ap2
File
Help
Tview
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
Source Files Present
radix.v1
radix.v1.pl
radix.v1.c
sorted.c
Cancel
OK
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

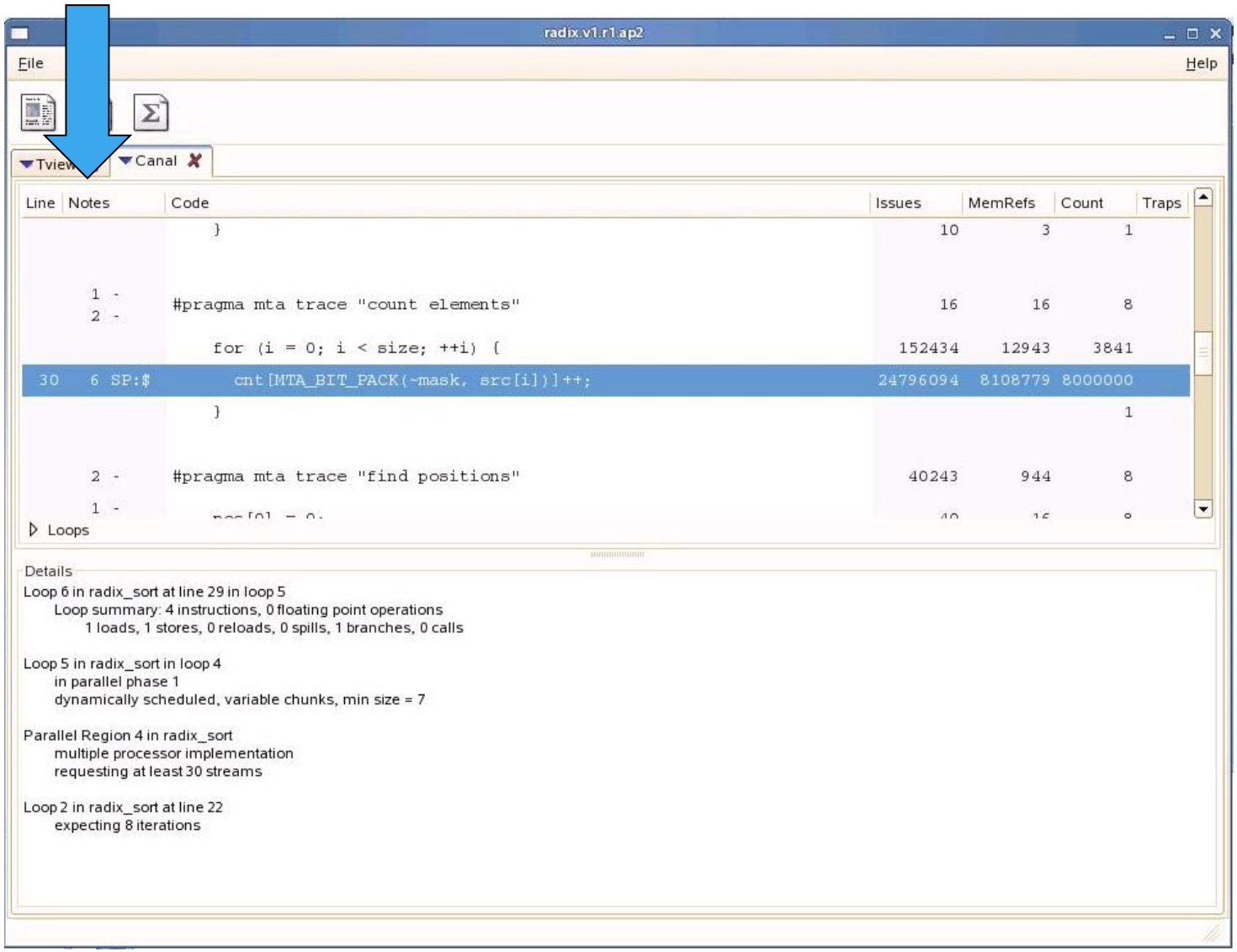

radix.v1.r1.ap2
File
Help
Tview
Canal
Line
Notes
Code
Issues
MemRefs
Count
Traps
#include "radix.h"
#include <stdio.h>
unsigned* radix_sort(unsigned* array, unsigned size) {
8650583 8250137 42
5
unsigned* src = array;
1 1 1
unsigned* dst = (unsigned*)malloc(size*sizeof(unsigned));
8 5 1
unsigned buckets = 1 << 8;
1
unsigned* cnt = (unsigned*)malloc(buckets*sizeof(unsigned));
259 256 1
Loops
Details

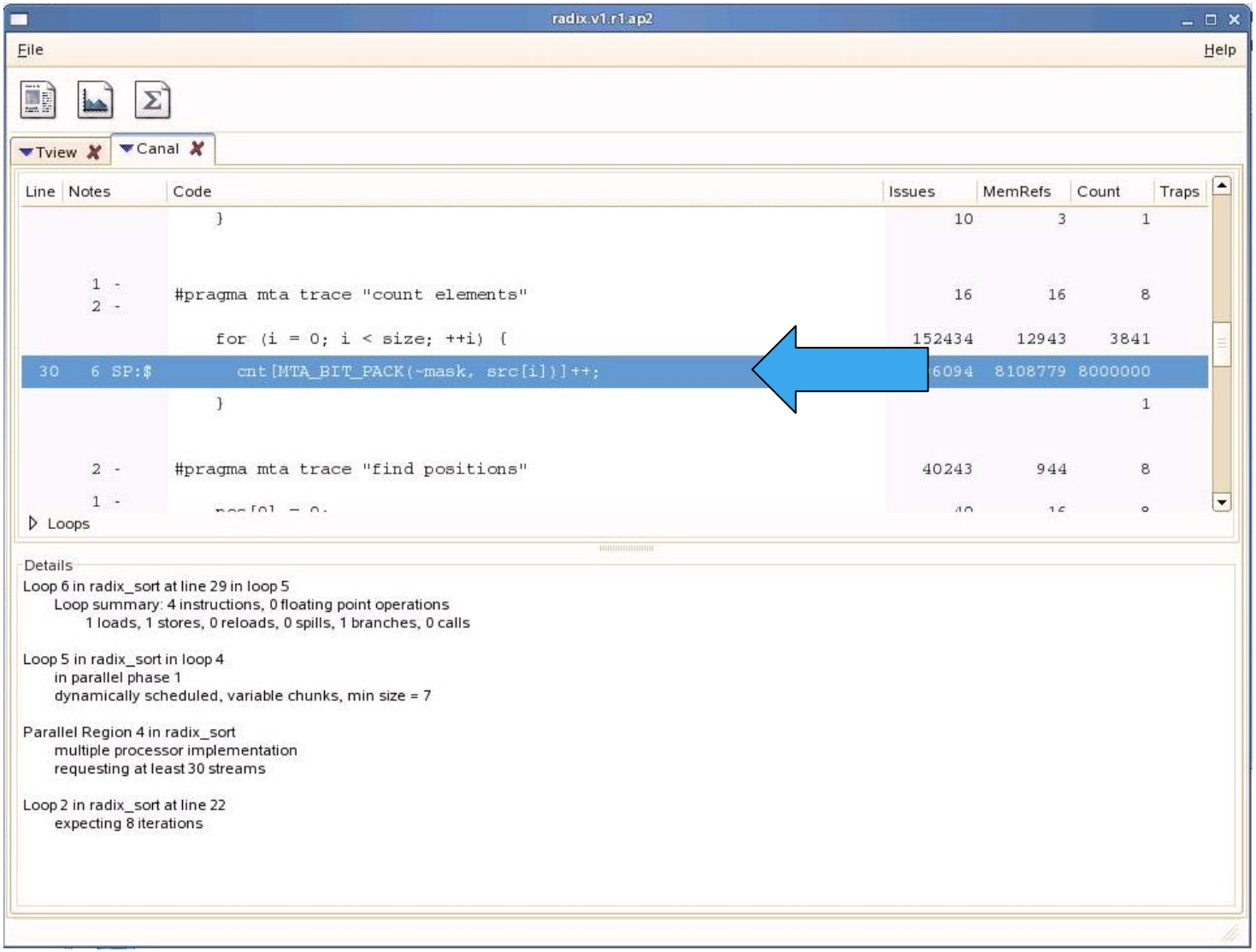

radix.v1.r1.ap2
File
Help
Tview
Canal
Line
Notes
Code
Issues
MemRefs
Count
Traps
#include "radix.h"
#include <stdio.h>
unsigned* radix_sort(unsigned* array, unsigned size) {
8650583
8250137
42
5
unsigned* src = array;
1
1
1
unsigned* dst = (unsigned*)malloc(size*sizeof(unsigned));
8
5
1
unsigned buckets = 1 << 8;
1
unsigned* cnt = (unsigned*)malloc(buckets*sizeof(unsigned));
259
256
1
Loops
Details

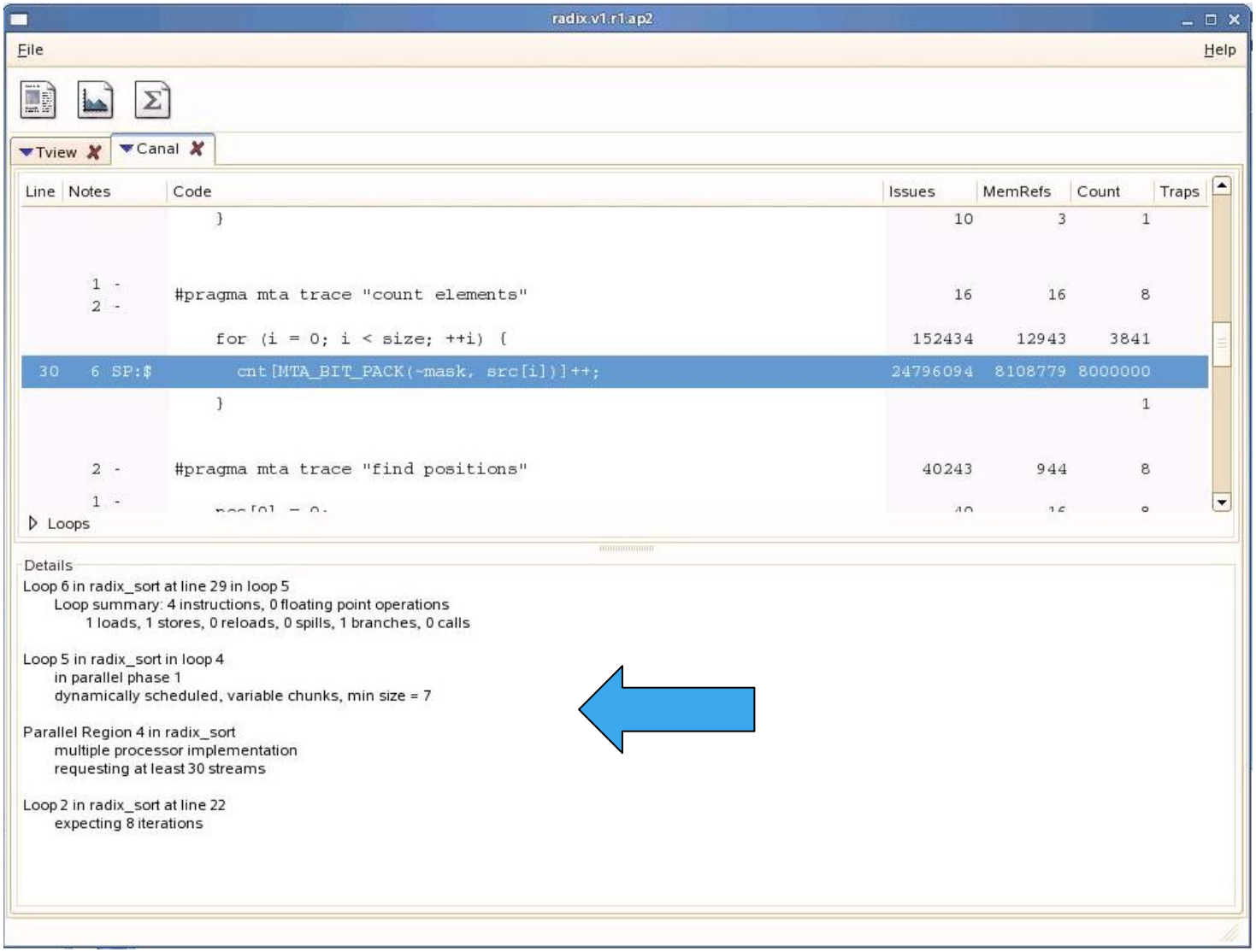

radix.v1.r1.ap2
File
Help
Tview
Canal
Line
Notes
Code
Issues
MemRefs
Count
Traps
#include "radix.h"
#include <stdio.h>
unsigned* radix_sort(unsigned* array, unsigned size) {
8650583 8250137 42
5
unsigned* src = array;
1 1 1
unsigned* dst = (unsigned*)malloc(size*sizeof(unsigned));
8 5 1
unsigned buckets = 1 << 8;
1
unsigned* cnt = (unsigned*)malloc(buckets*sizeof(unsigned));
259 256 1
Loops
Details

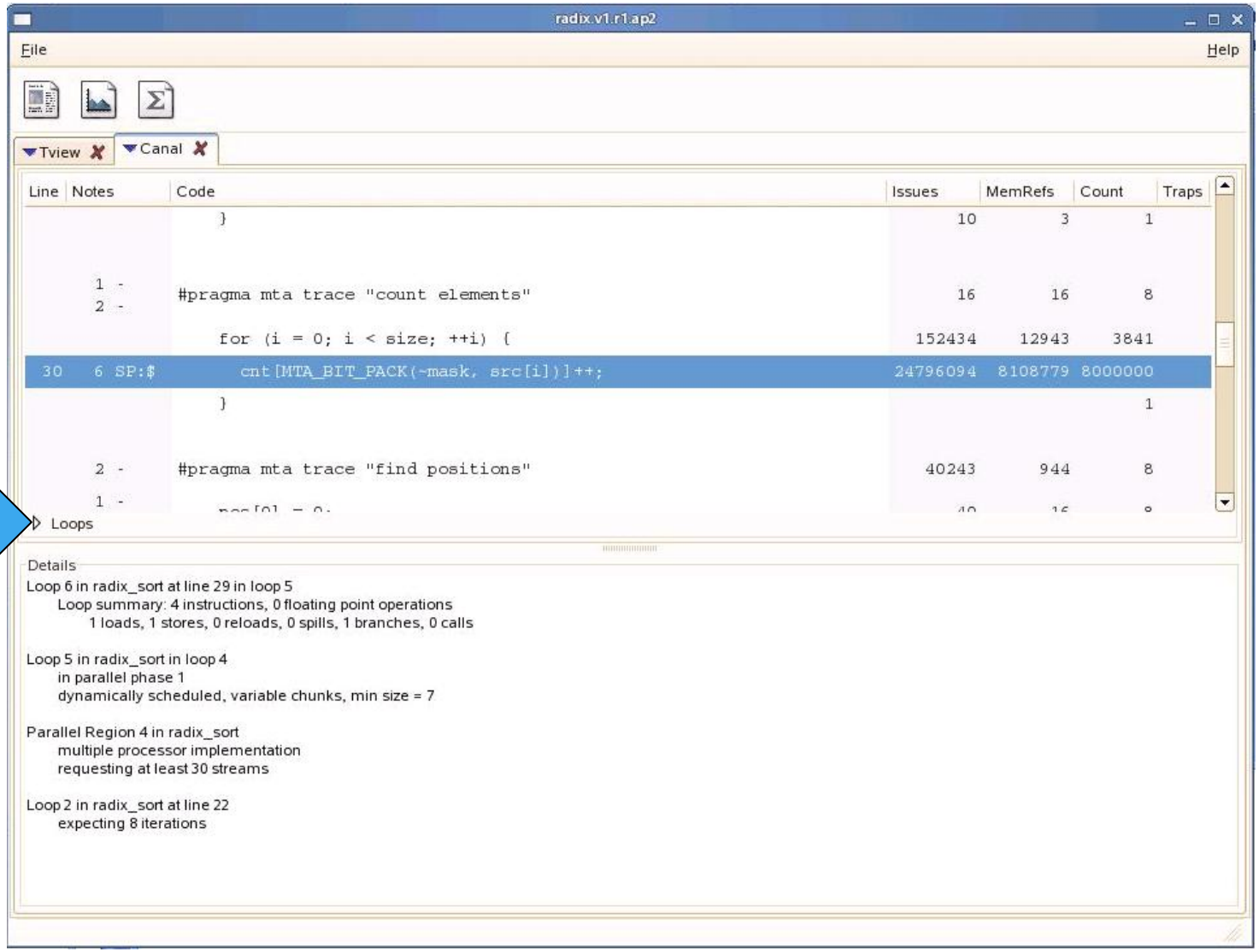

radix.v1.r1.ap2
File
Help
Tview
Select Source...
Toolbars
Columns
Change Font...
Panel Actions
Panel Help
Line
Notes
Issues
MemRefs
Count
Traps
10 3 1
1 - trace "count elements" 16 16 8
2 - 0; i < size; ++i) { 152434 12943 3841
30 6 SI: A_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations

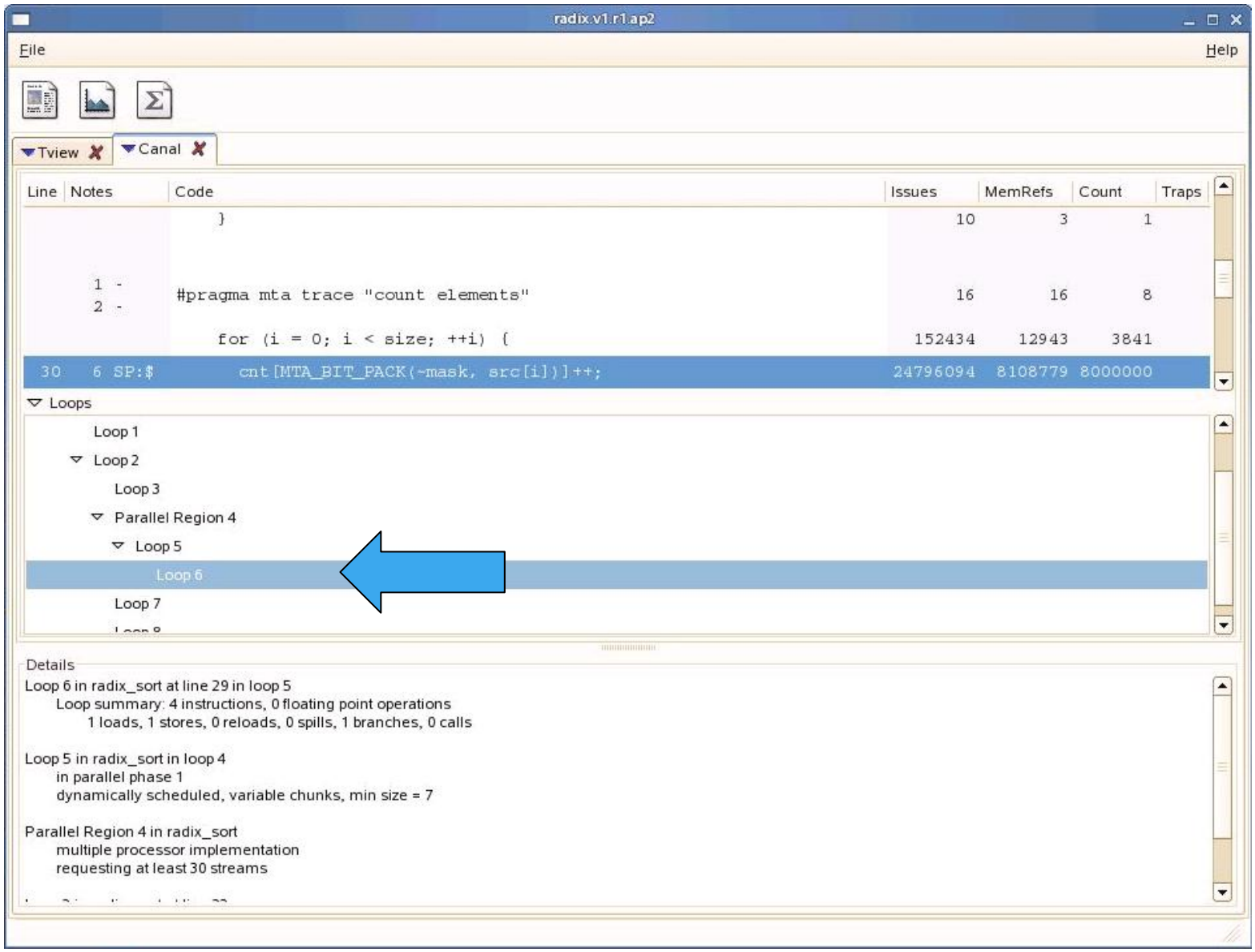

radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 -
#pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
} 1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations


radix.v1.r1.ap2
File
Help
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 - #pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
} 1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations


radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 -
#pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 6094 8108779 8000000
}
1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations


radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 - #pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
} 1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations

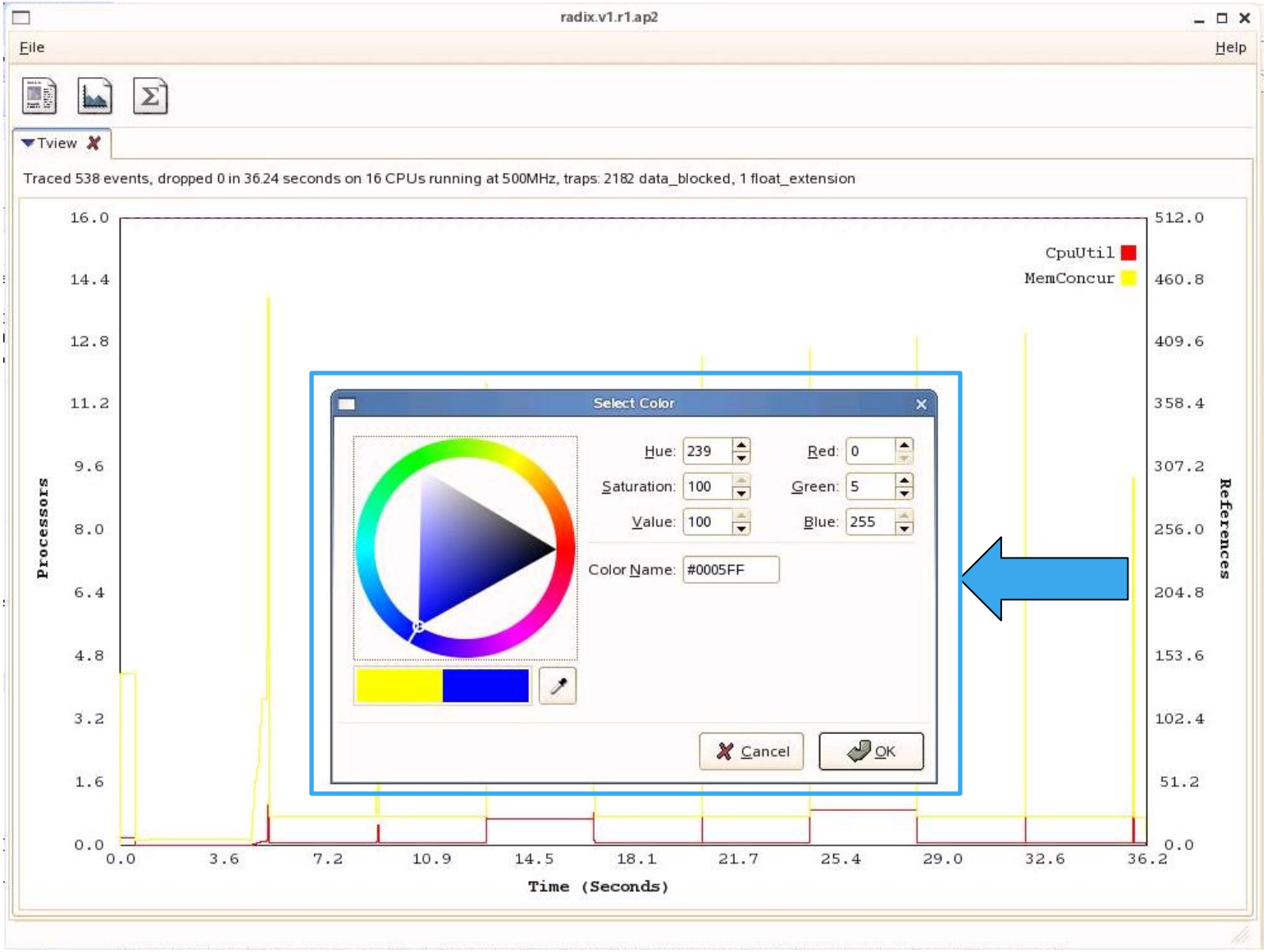

radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 -
#pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
} 1
2 - #pragma mta trace "find positions" 40243 944 8
1 -
Loops
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams
Loop 2 in radix_sort at line 22
expecting 8 iterations

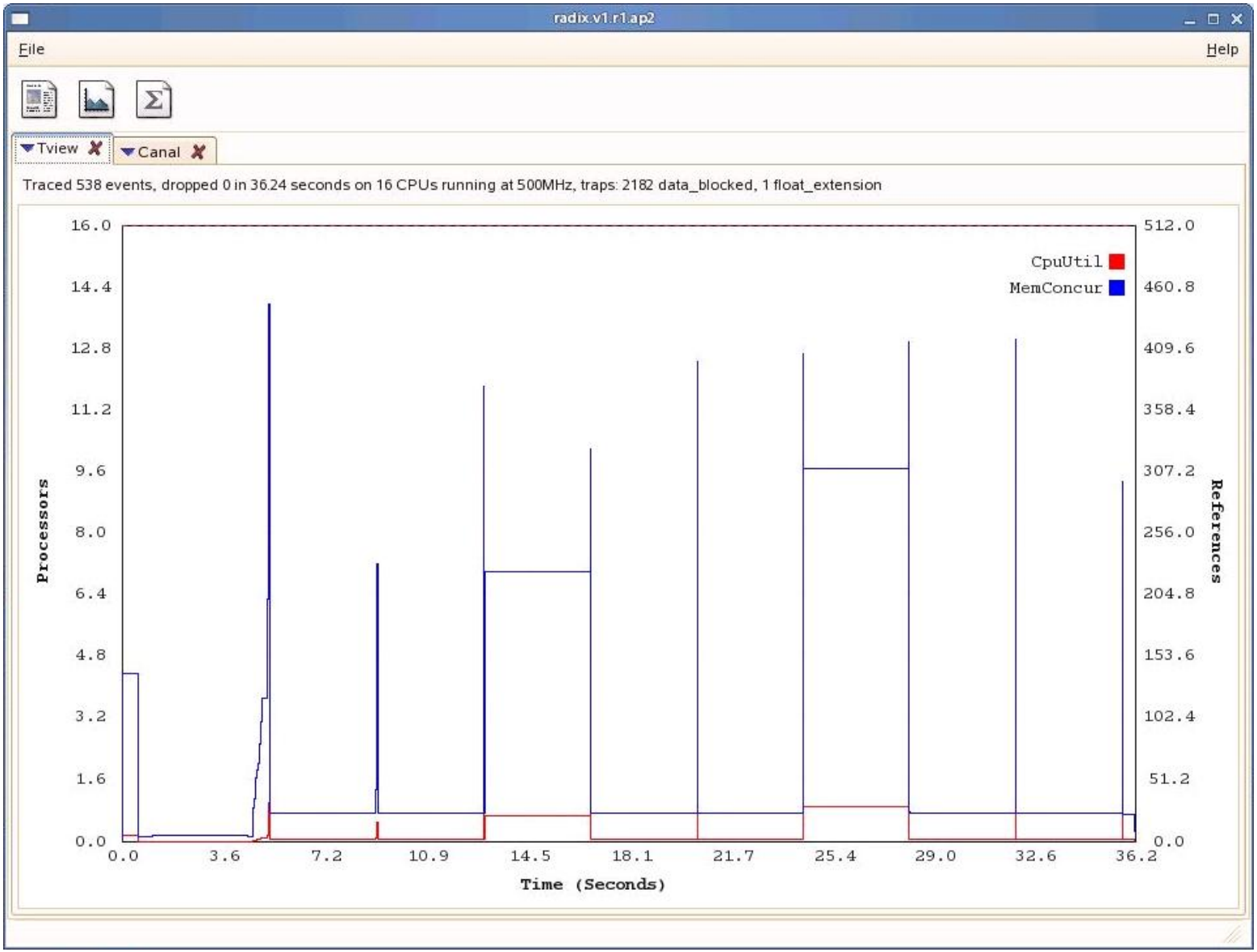

radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
} 10 3 1
1 -
2 -
#pragma mta trace "count elements" 16 16 8
for (i = 0; i < size; ++i) { 152434 12943 3841
30 6 SP:$ cnt[MTA_BIT_PACK(~mask, src[i])]++; 24796094 8108779 8000000
Loops
Loop 1
Loop 2
Loop 3
Parallel Region 4
Loop 5
Loop 6
Loop 7
Details
Loop 6 in radix_sort at line 29 in loop 5
Loop summary: 4 instructions, 0 floating point operations
1 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 5 in radix_sort in loop 4
in parallel phase 1
dynamically scheduled, variable chunks, min size = 7
Parallel Region 4 in radix_sort
multiple processor implementation
requesting at least 30 streams

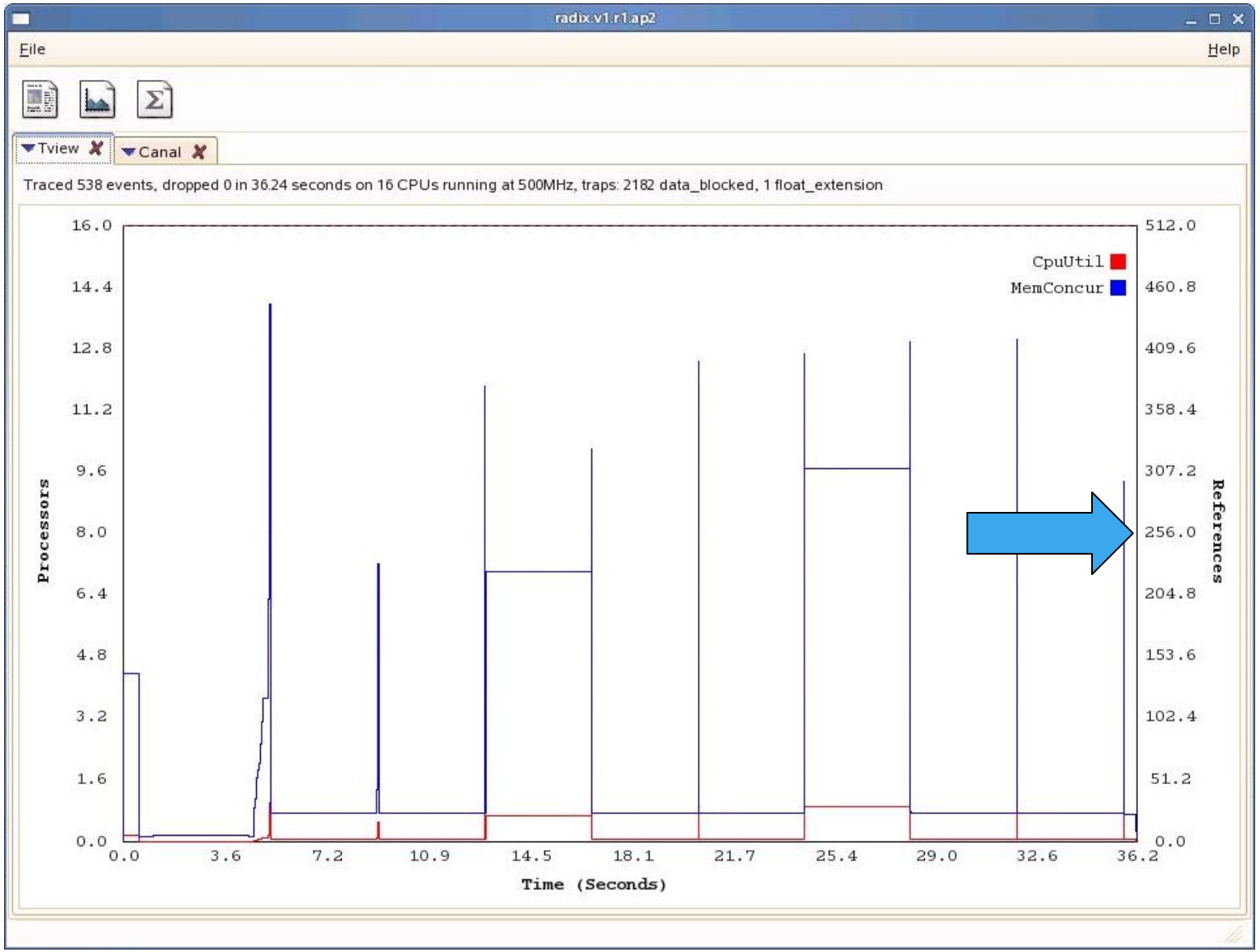

File
Help
radix.v1.r1.ap2
Tview
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

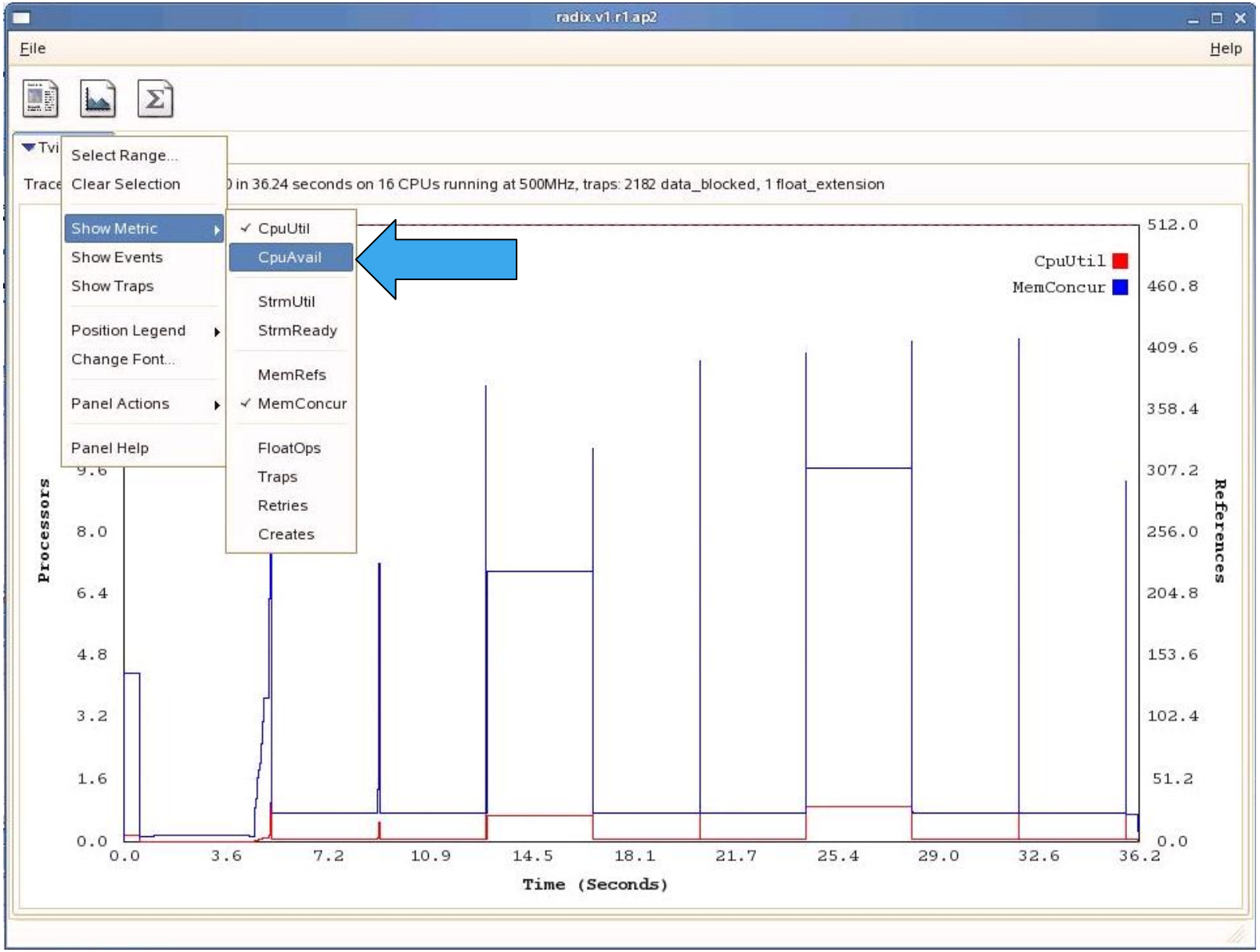

File
Help
radix.v1.r1.ap2
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182
1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

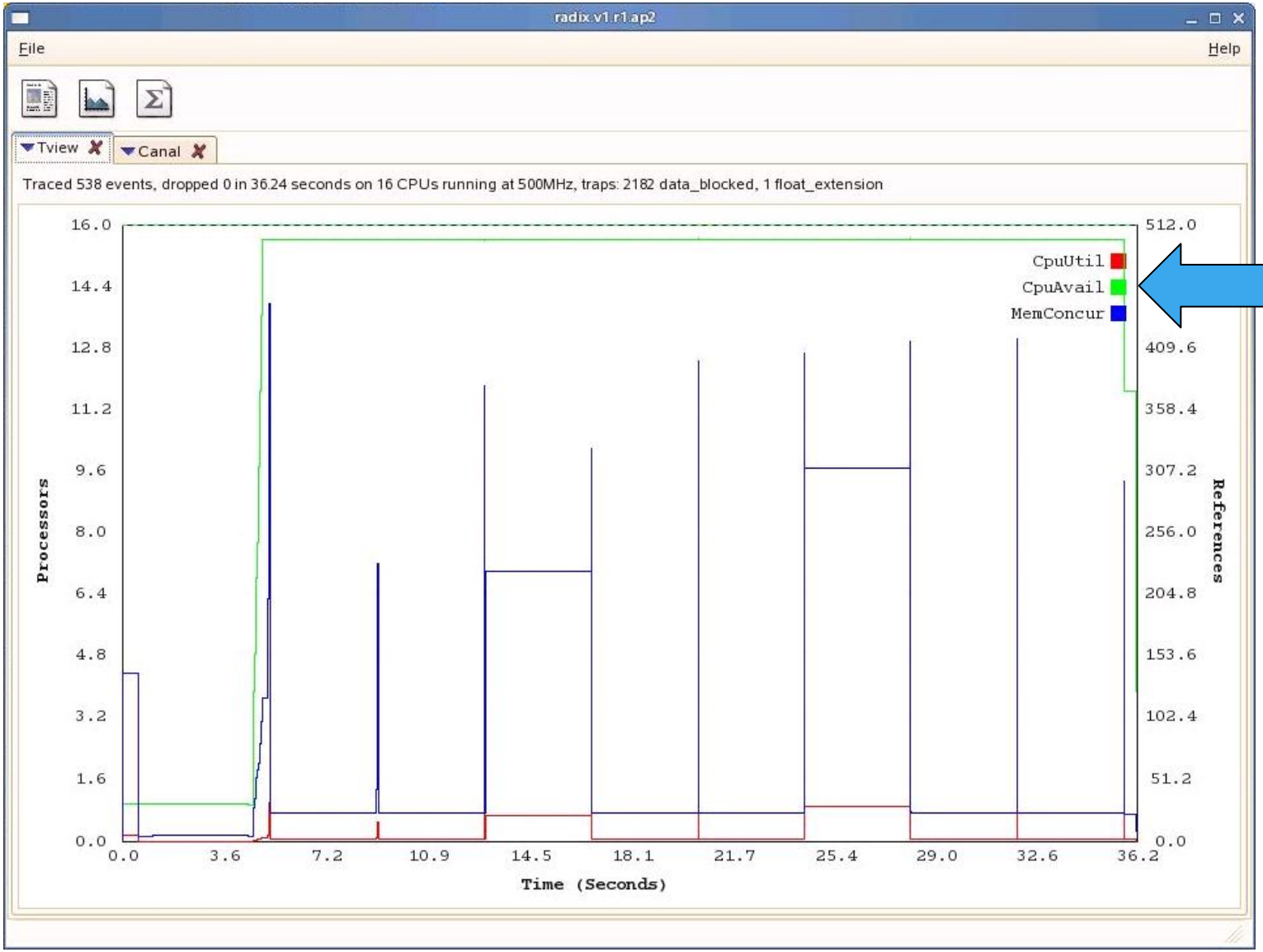

File
Help
radix.v1.r1.ap2
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
References
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)

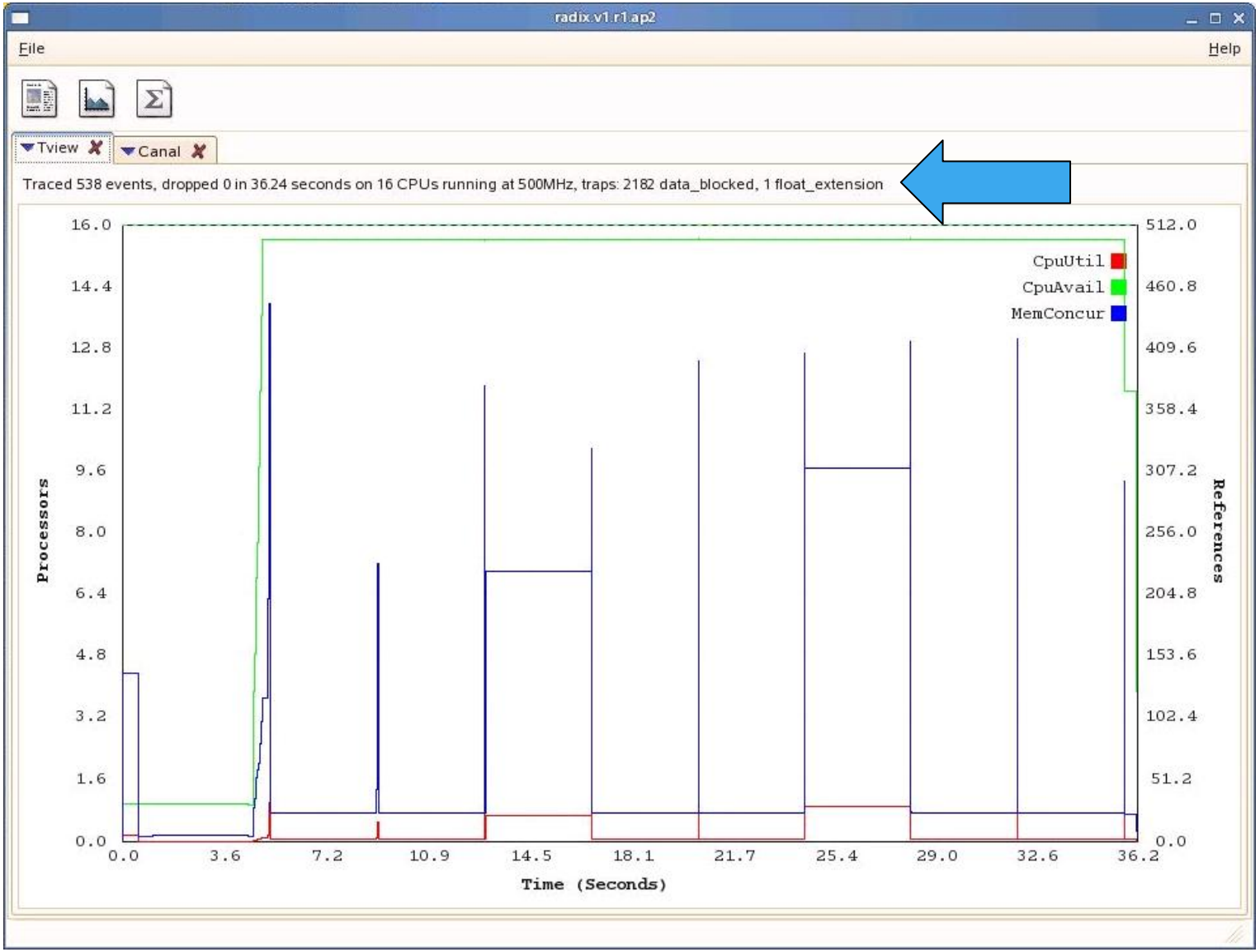

radix.v1.r1.ap2
File
Help
Tview
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Select Color
Hue: 239
Saturation: 100
Value: 100
Red: 0
Green: 5
Blue: 255
Color Name: #0005FF
Cancel
OK

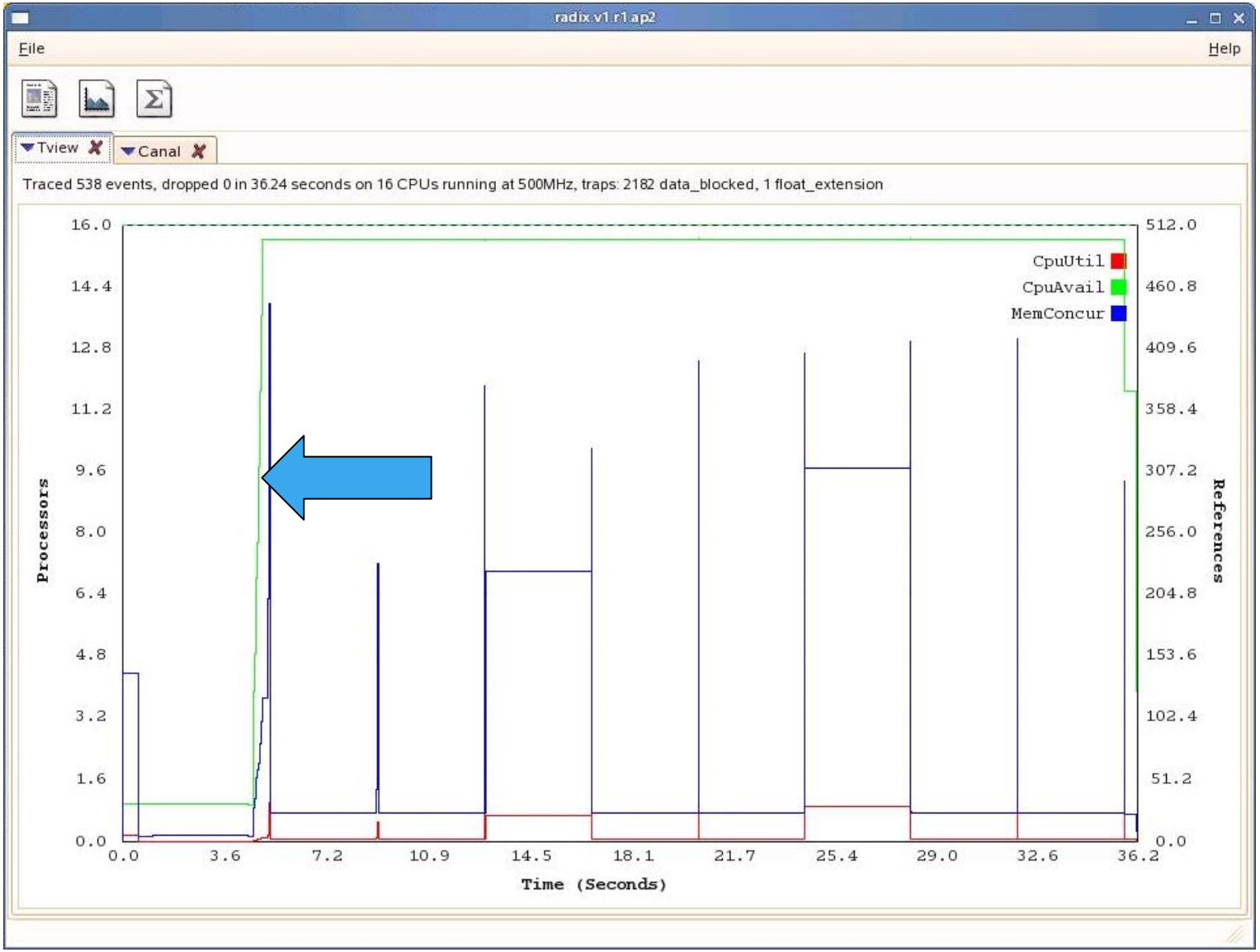

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2


radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

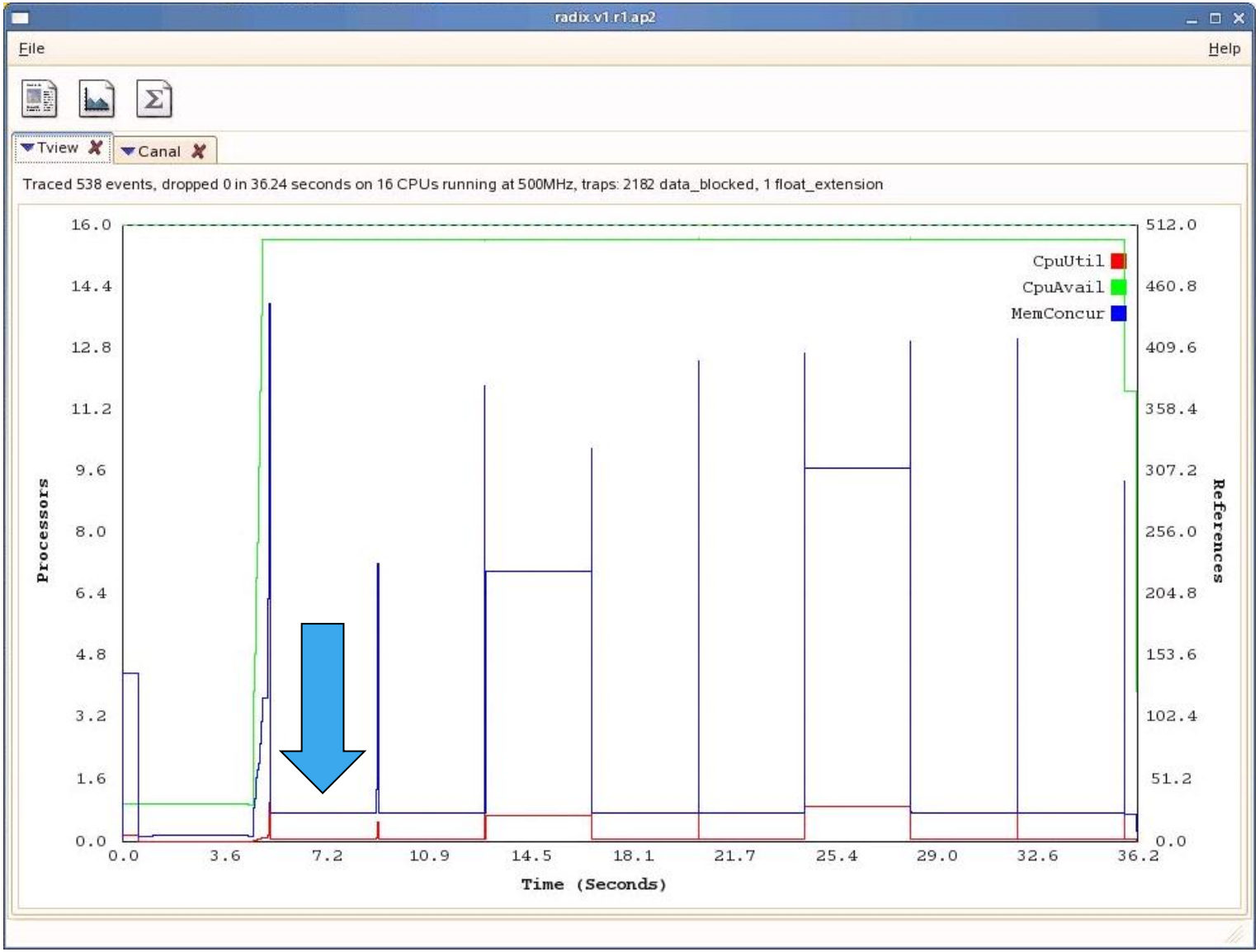

radix.v1.r1.ap2
File
Help
Select Range...
Clear Selection
Show Metric
Show Events
Show Traps
Position Legend
Change Font...
Panel Actions
Panel Help
CpuUtil
CpuAvail
StrmUtil
StrmReady
MemRefs
MemConcur
FloatOps
Traps
Retries
Creates
in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
MemConcur
Processors
References
Time (Seconds)
0.0
1.6
3.2
4.8
6.4
8.0
9.6
51.2
102.4
153.6
204.8
256.0
307.2
358.4
409.6
460.8
512.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

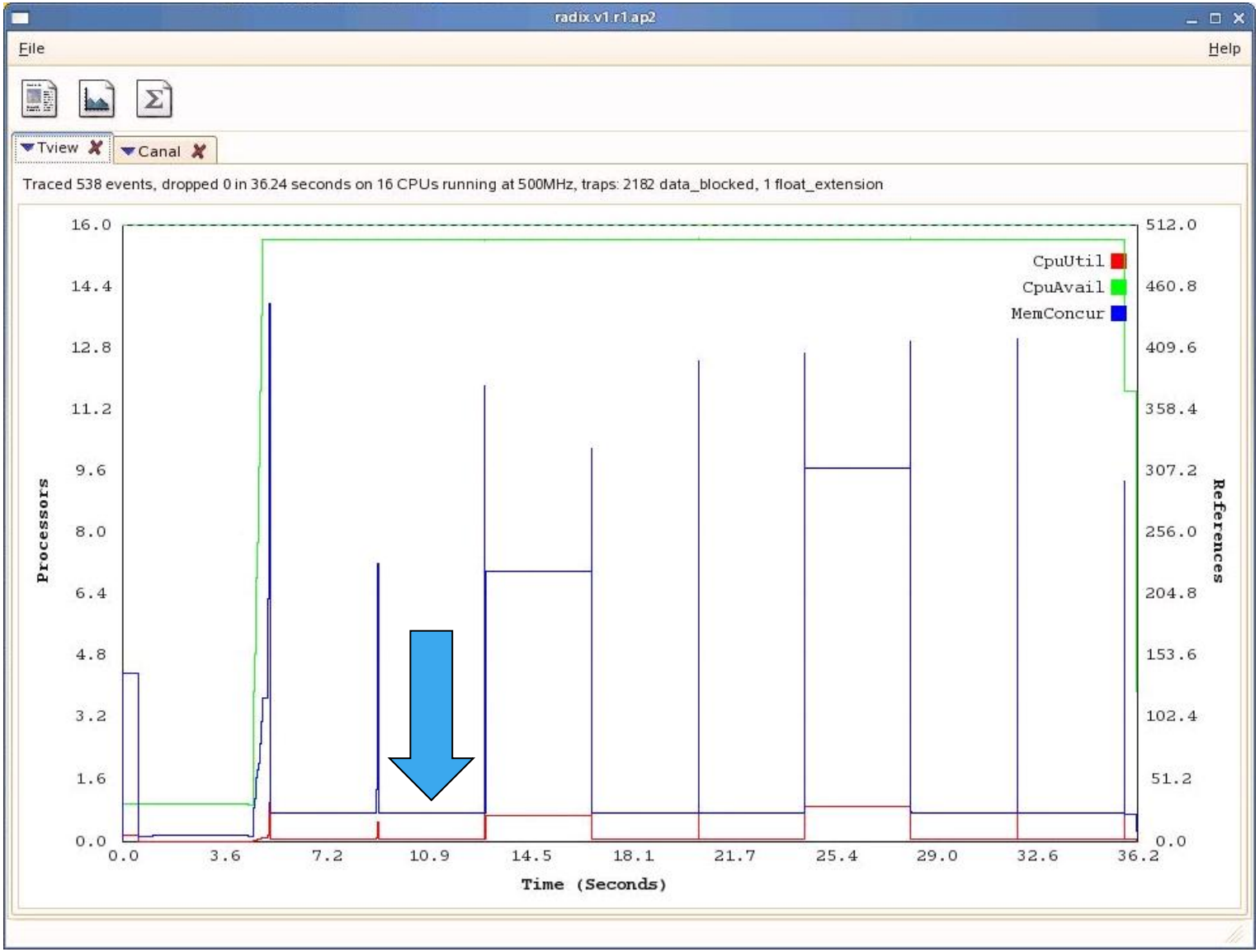

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

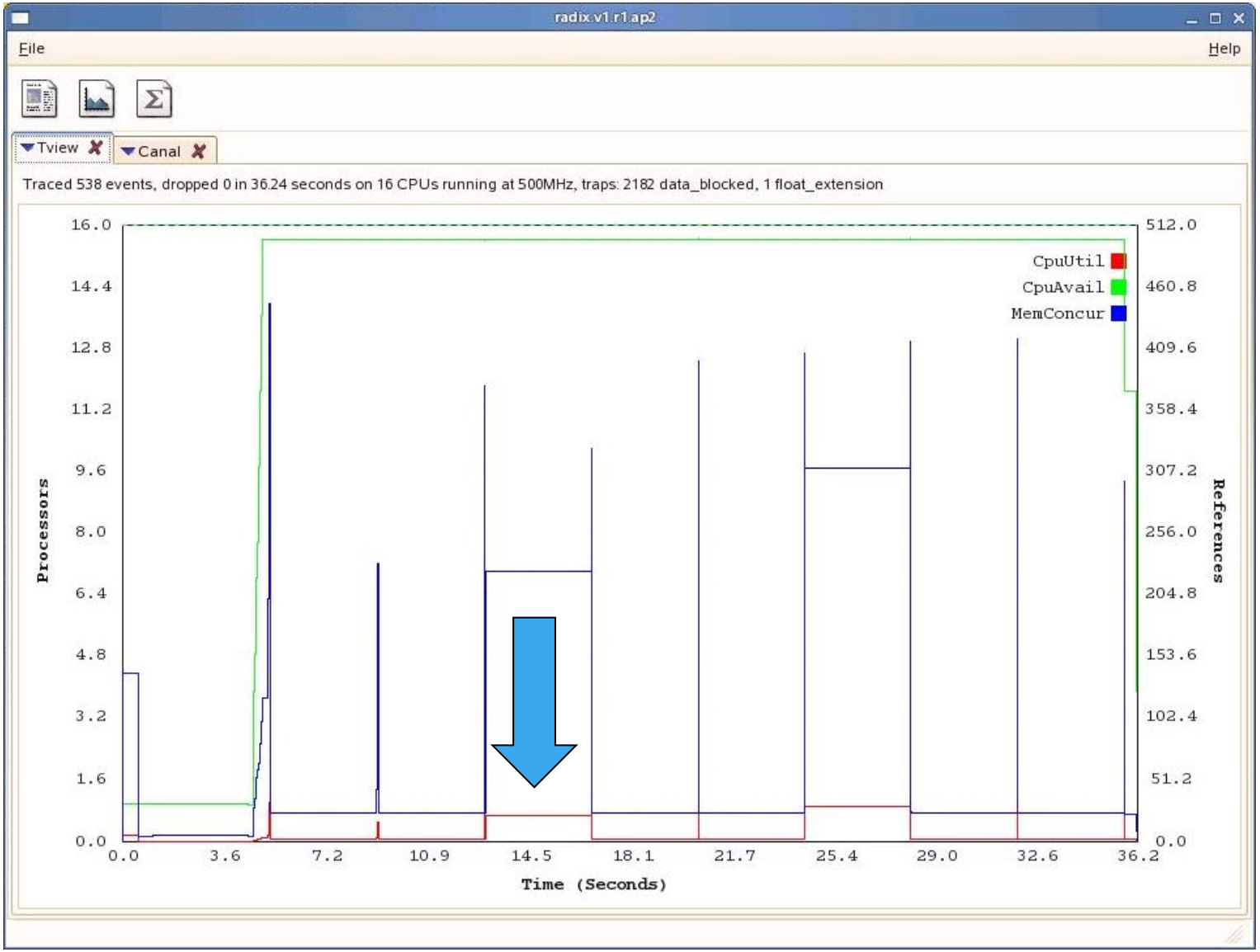

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

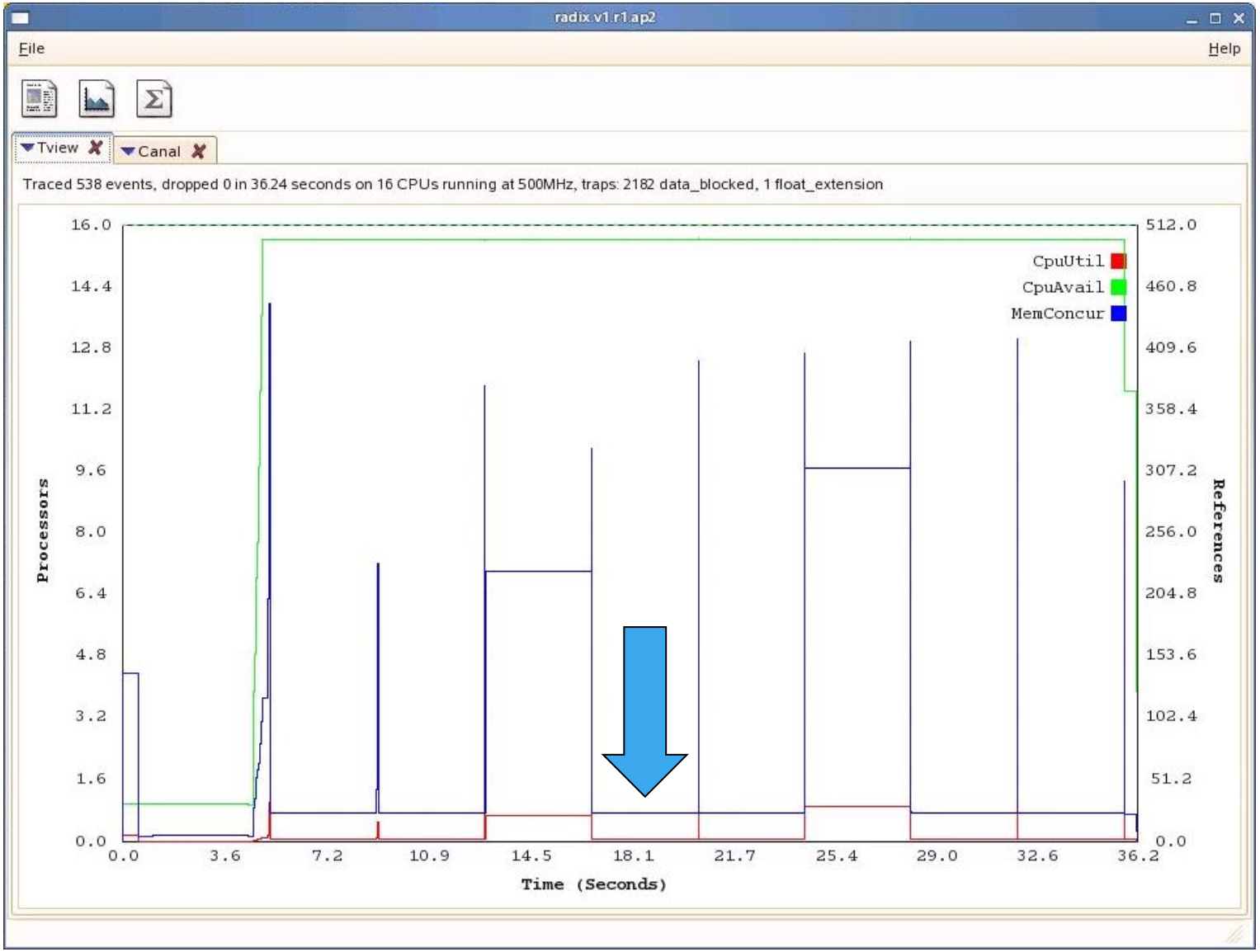

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

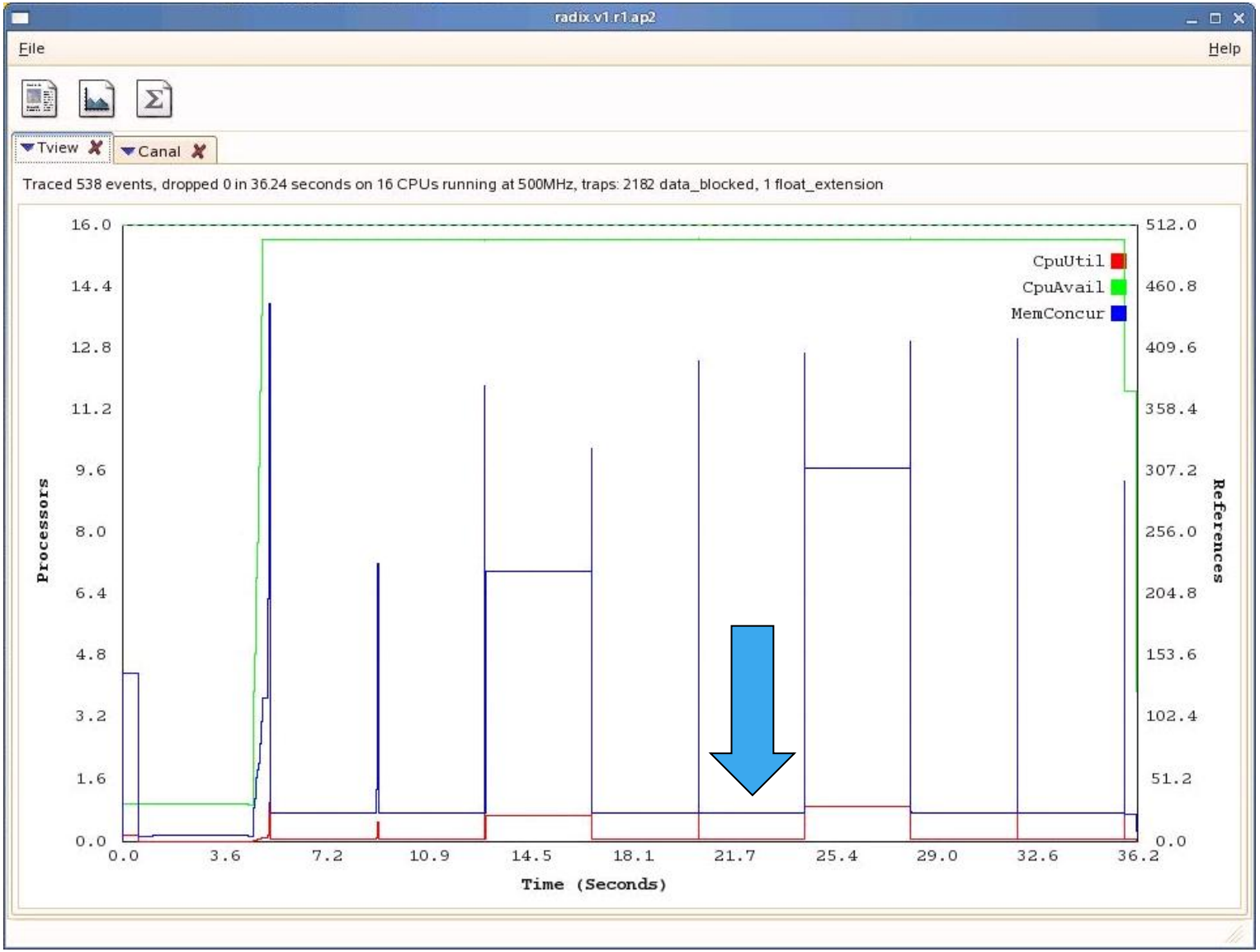

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

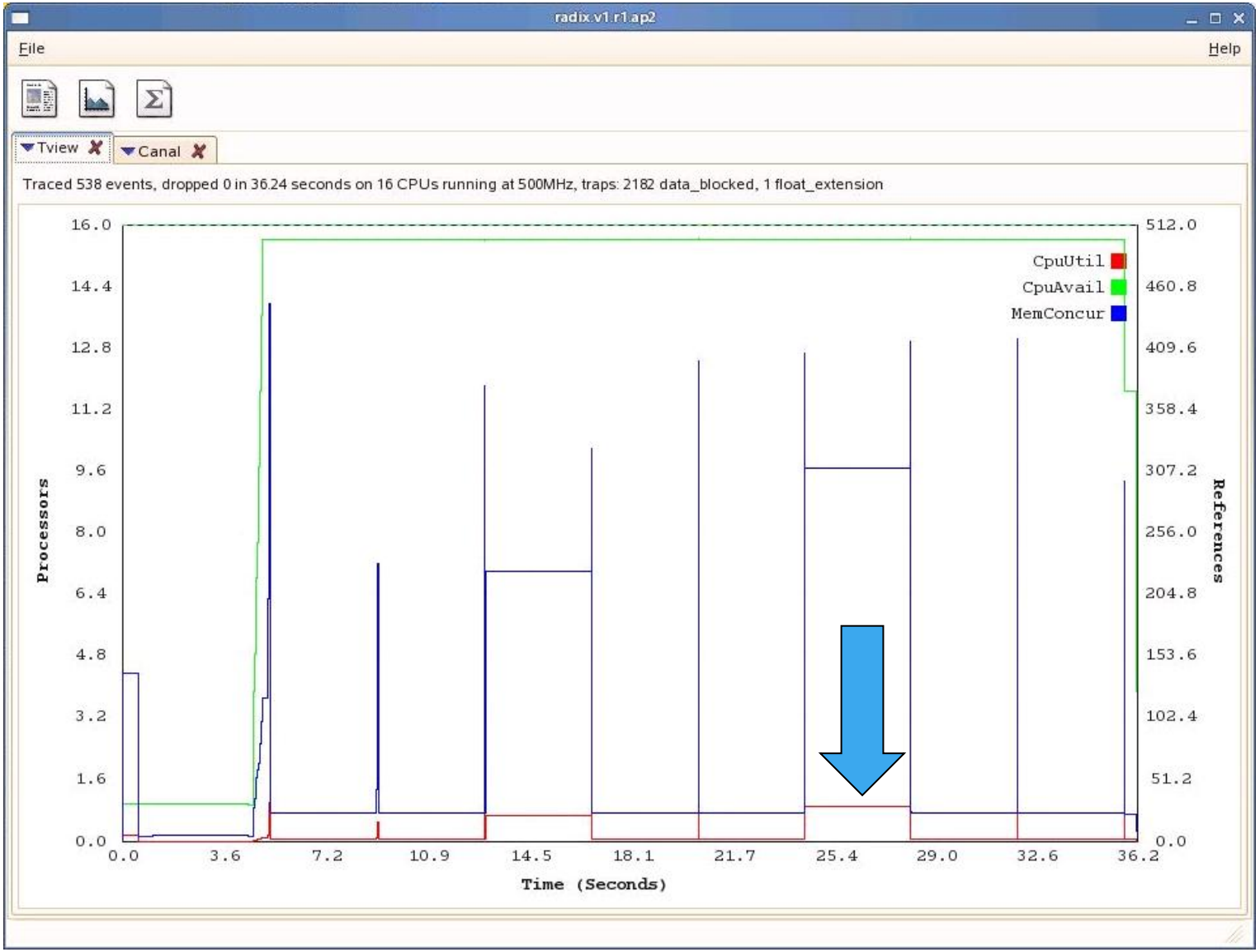

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

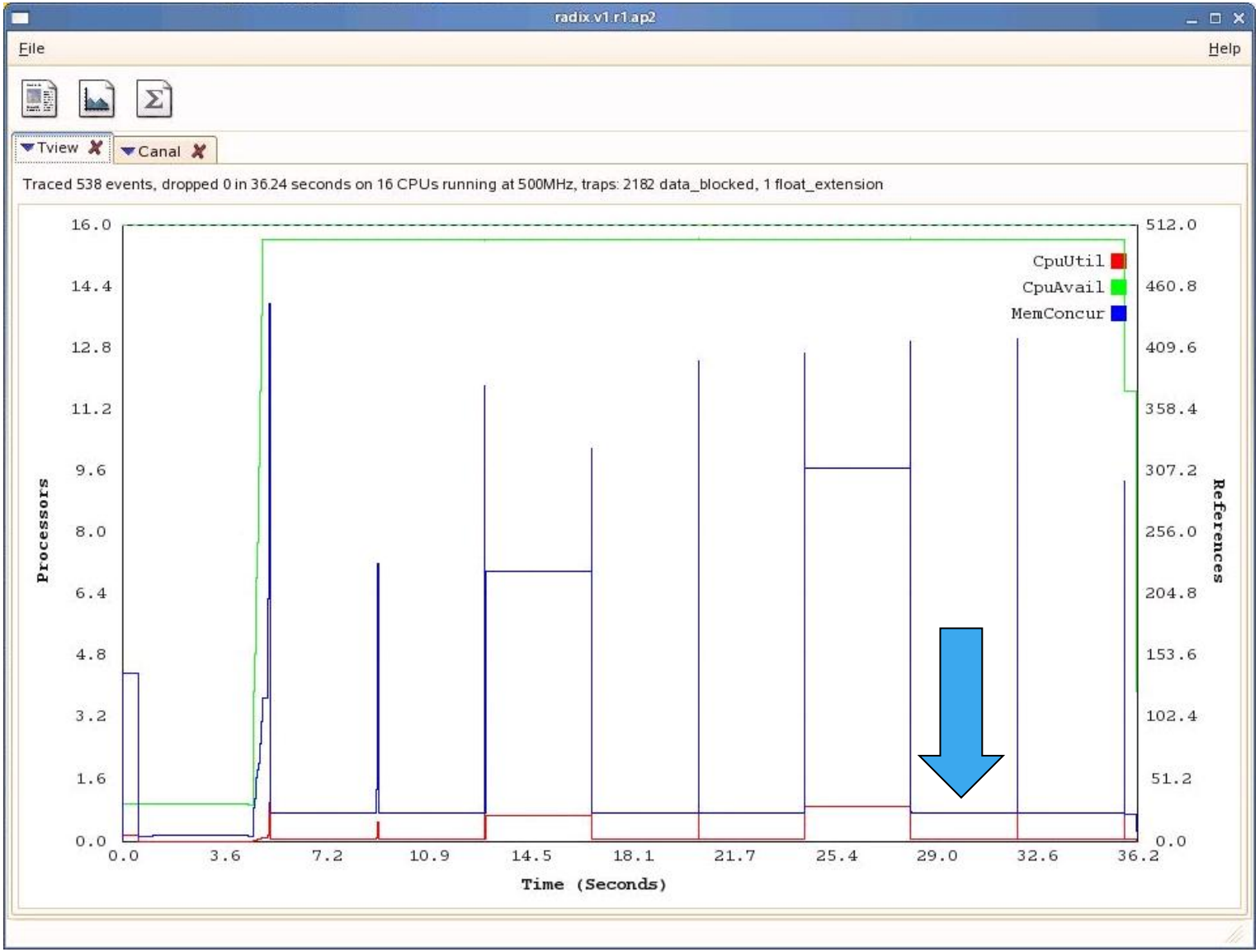

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

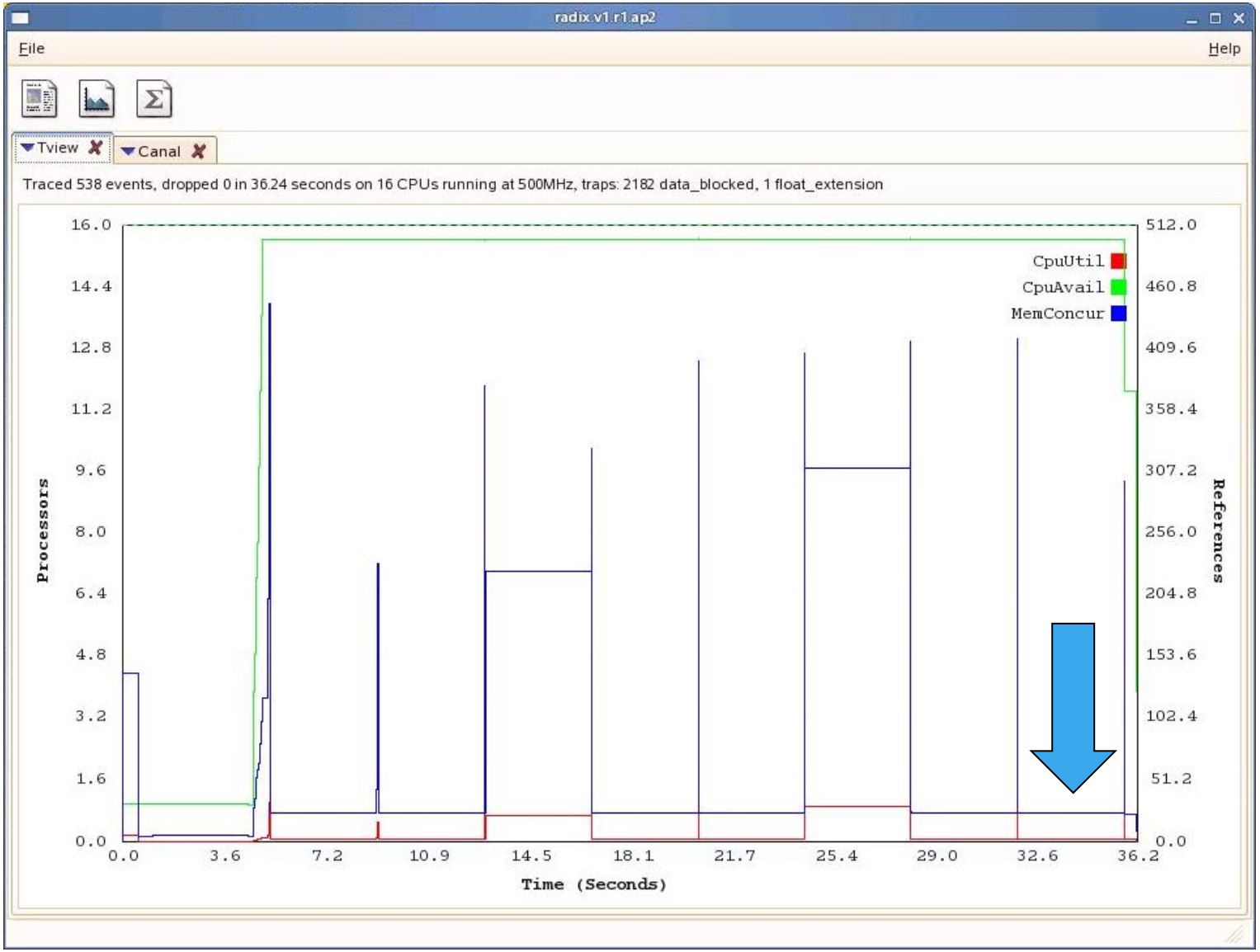

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

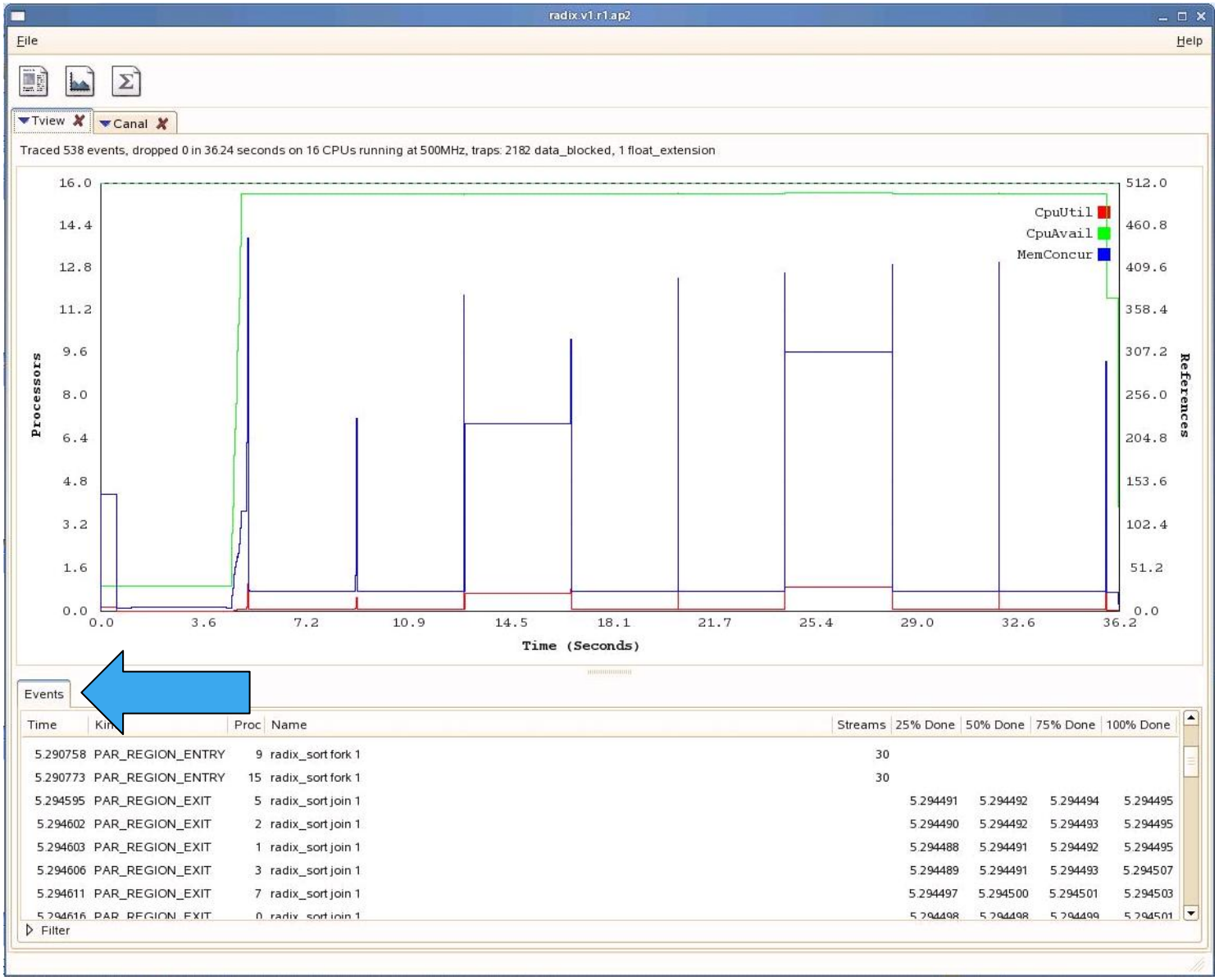

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

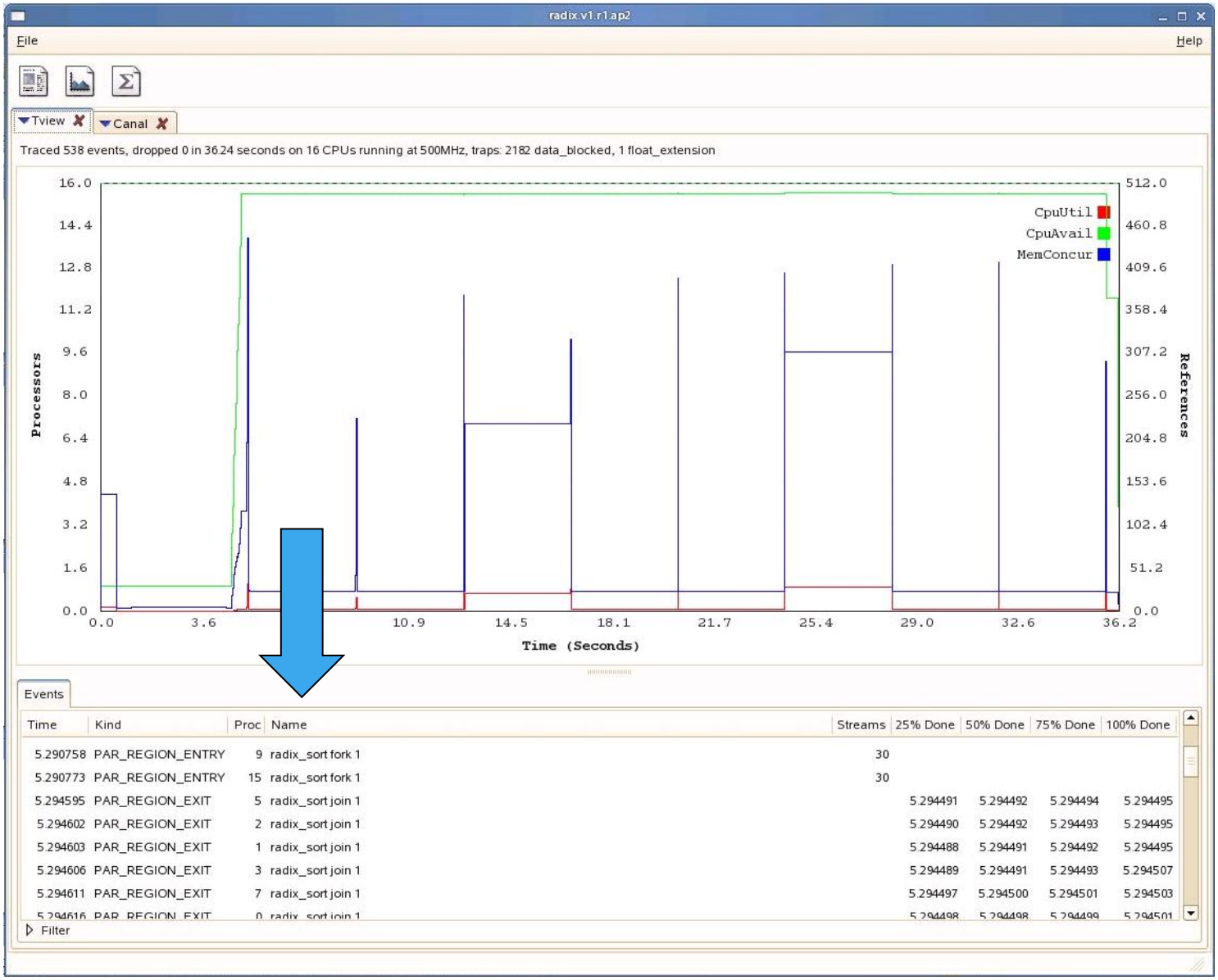

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

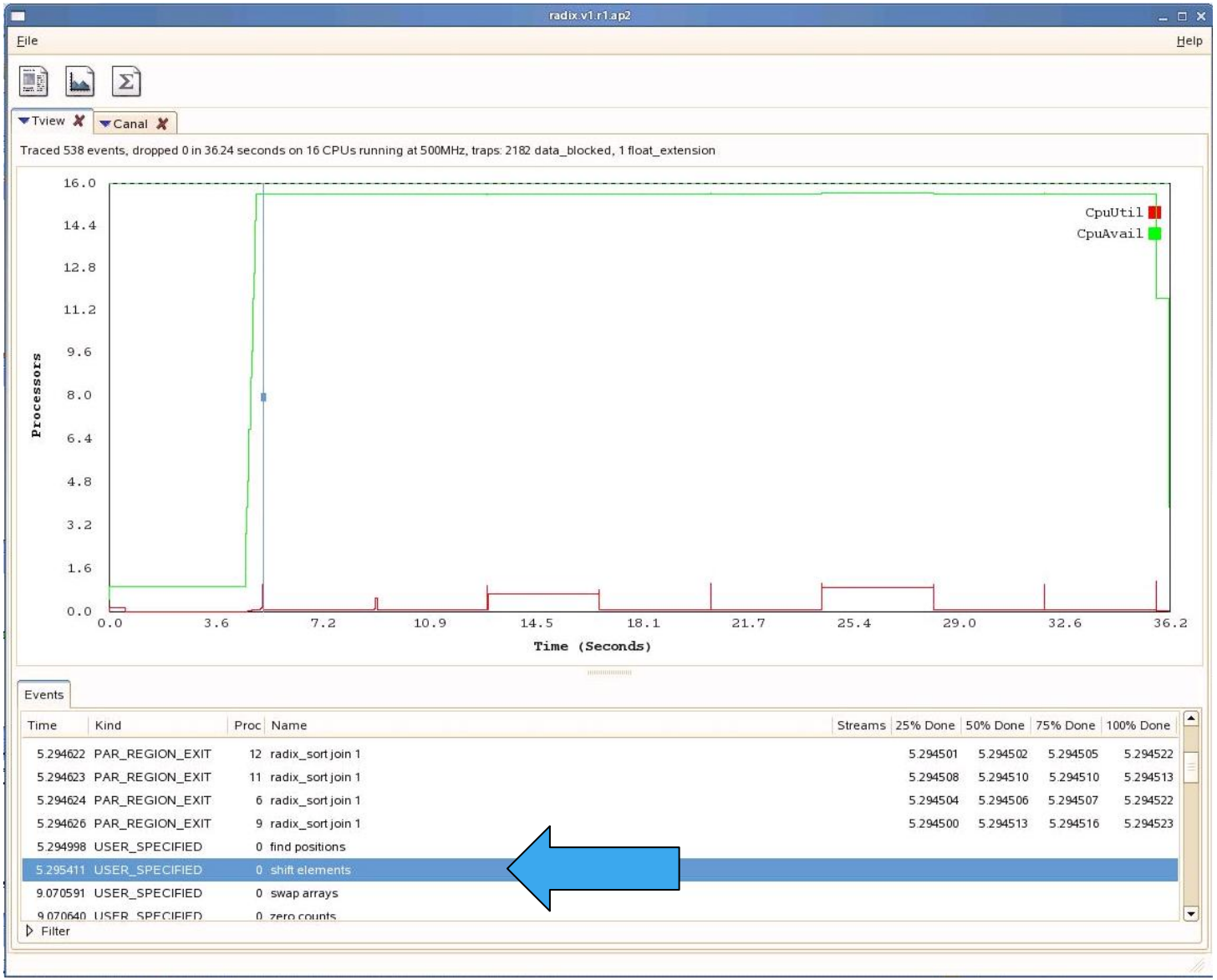

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

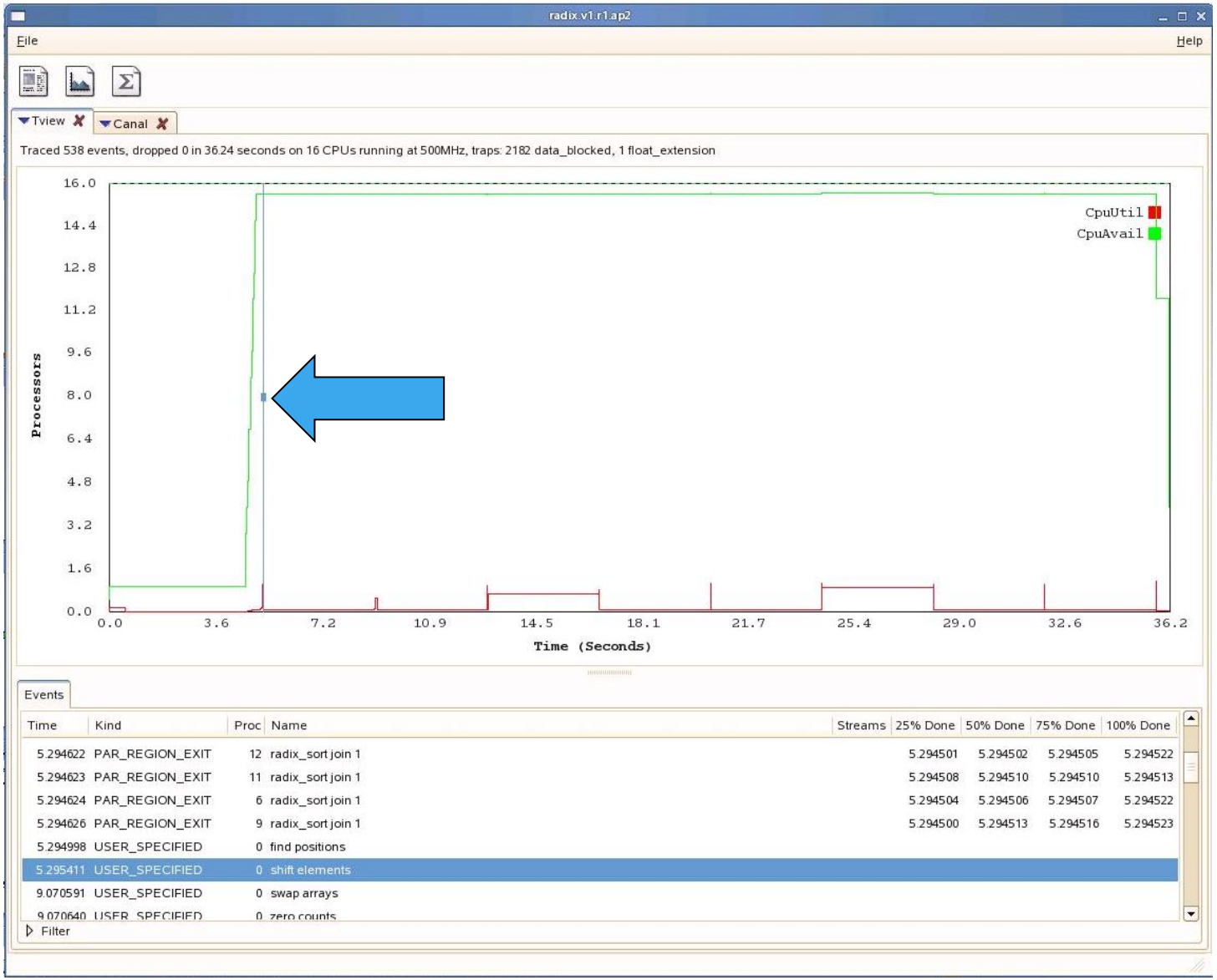

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

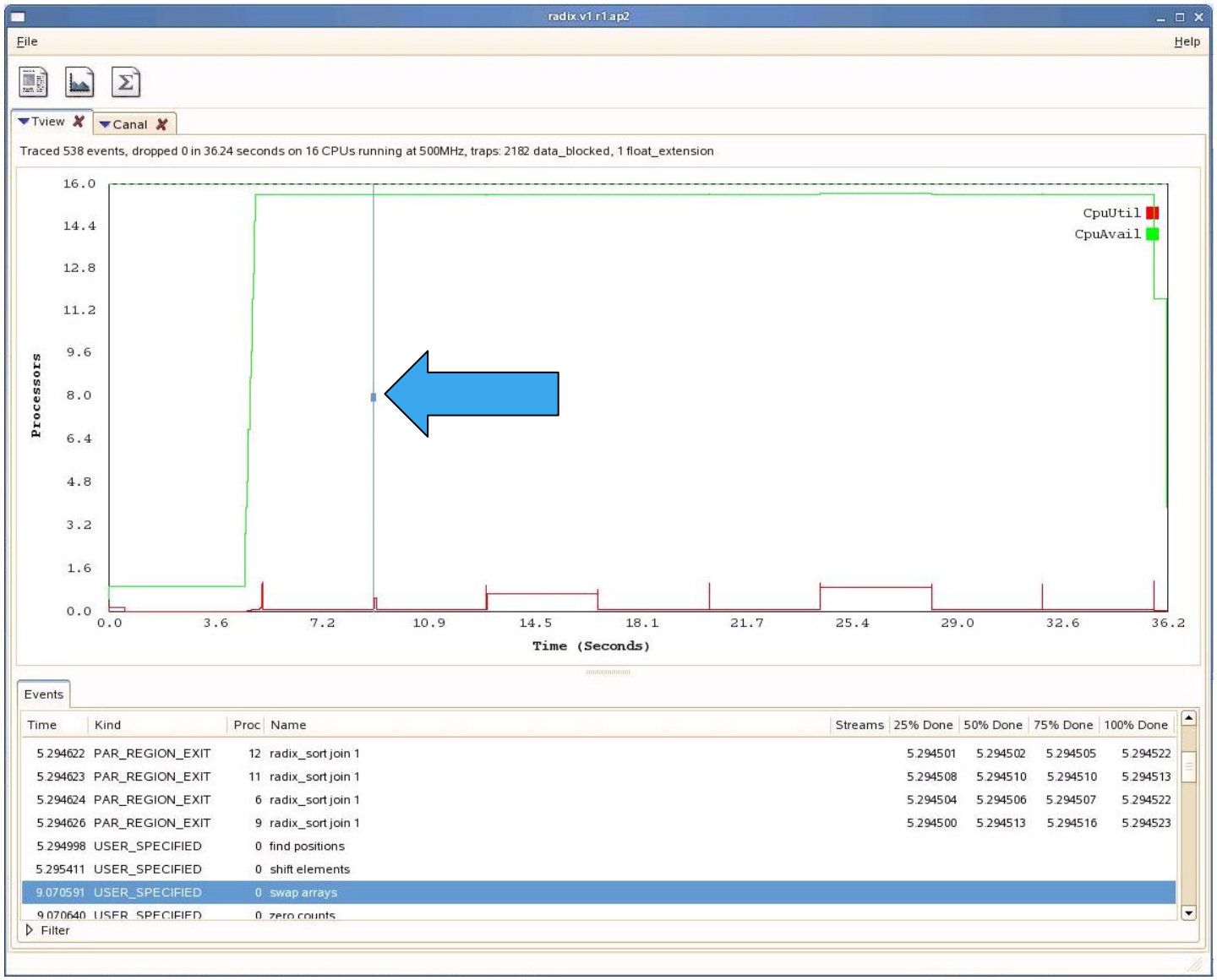

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2

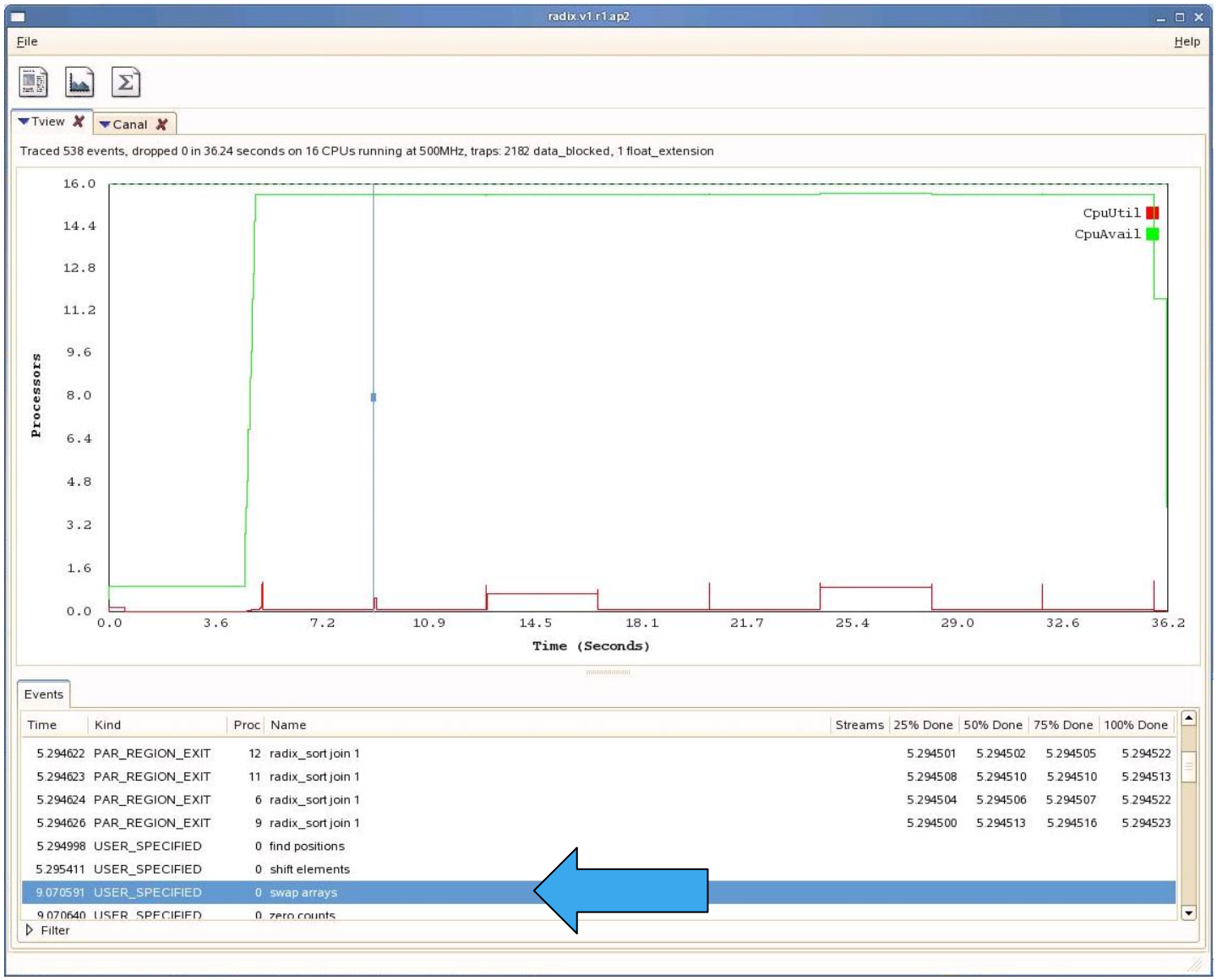

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
References
CpuUtil
CpuAvail
MemConcur
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time
Kin
Proc
Name
Streams
25% Done
50% Done
75% Done
100% Done
5.290758 PAR_REGION_ENTRY 9 radix_sort fork 1 30
5.290773 PAR_REGION_ENTRY 15 radix_sort fork 1 30
5.294595 PAR_REGION_EXIT 5 radix_sort join 1 5.294491 5.294492 5.294494 5.294495
5.294602 PAR_REGION_EXIT 2 radix_sort join 1 5.294490 5.294492 5.294493 5.294495
5.294603 PAR_REGION_EXIT 1 radix_sort join 1 5.294488 5.294491 5.294492 5.294495
5.294606 PAR_REGION_EXIT 3 radix_sort join 1 5.294489 5.294491 5.294493 5.294507
5.294611 PAR_REGION_EXIT 7 radix_sort join 1 5.294497 5.294500 5.294501 5.294503
Filter

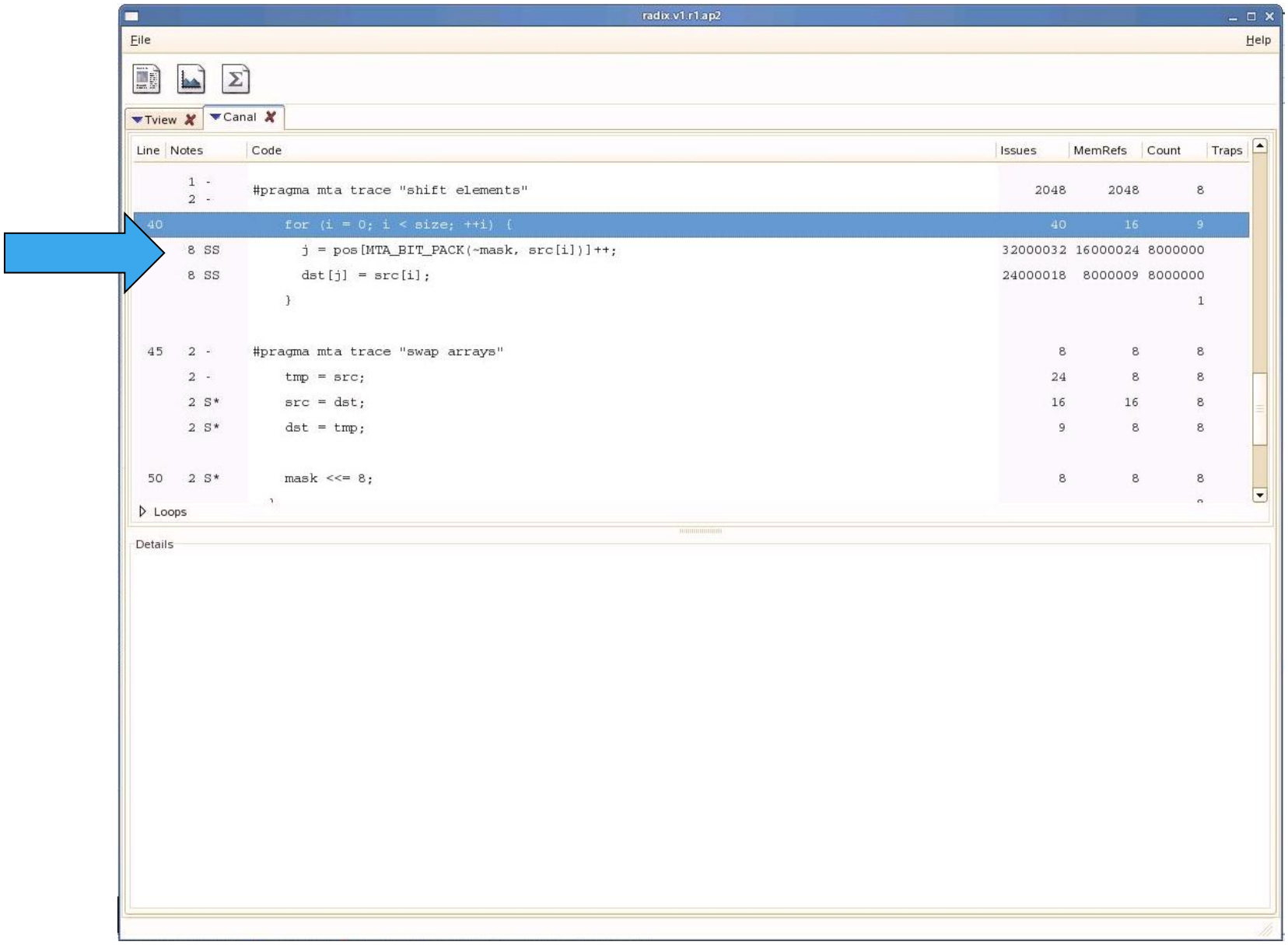

radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
512.0
460.8
409.6
358.4
307.2
256.0
204.8
153.6
102.4
51.2
0.0
References
CpuUtil
CpuAvail
MemConcur
0.0
3.6
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time Kind Proc Name Streams 25% Done 50% Done 75% Done 100% Done
5.290758 PAR_REGION_ENTRY 9 radix_sort fork 1 30
5.290773 PAR_REGION_ENTRY 15 radix_sort fork 1 30
5.294595 PAR_REGION_EXIT 5 radix_sort join 1 5.294491 5.294492 5.294494 5.294495
5.294602 PAR_REGION_EXIT 2 radix_sort join 1 5.294490 5.294492 5.294493 5.294495
5.294603 PAR_REGION_EXIT 1 radix_sort join 1 5.294488 5.294491 5.294492 5.294495
5.294606 PAR_REGION_EXIT 3 radix_sort join 1 5.294489 5.294491 5.294493 5.294507
5.294611 PAR_REGION_EXIT 7 radix_sort join 1 5.294497 5.294500 5.294501 5.294503
Filter


radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
CpuUtil
CpuAvail
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time Kind Proc Name Streams 25% Done 50% Done 75% Done 100% Done
5.294622 PAR_REGION_EXIT 12 radix_sort join 1 5.294501 5.294502 5.294505 5.294522
5.294623 PAR_REGION_EXIT 11 radix_sort join 1 5.294508 5.294510 5.294510 5.294513
5.294624 PAR_REGION_EXIT 6 radix_sort join 1 5.294504 5.294506 5.294507 5.294522
5.294626 PAR_REGION_EXIT 9 radix_sort join 1 5.294500 5.294513 5.294516 5.294523
5.294998 USER_SPECIFIED 0 find positions
5.295411 USER_SPECIFIED 0 shift elements
9.070591 USER_SPECIFIED 0 swap arrays
9.070640 USER_SPECIFIED 0 zero counts
Filter


radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
CpuUtil
CpuAvail
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time Kind Proc Name Streams 25% Done 50% Done 75% Done 100% Done
5.294622 PAR_REGION_EXIT 12 radix_sort join 1 5.294501 5.294502 5.294505 5.294522
5.294623 PAR_REGION_EXIT 11 radix_sort join 1 5.294508 5.294510 5.294510 5.294513
5.294624 PAR_REGION_EXIT 6 radix_sort join 1 5.294504 5.294506 5.294507 5.294522
5.294626 PAR_REGION_EXIT 9 radix_sort join 1 5.294500 5.294513 5.294516 5.294523
5.294998 USER_SPECIFIED 0 find positions
5.295411 USER_SPECIFIED 0 shift elements
9.070591 USER_SPECIFIED 0 swap arrays
9.070640 USER_SPECIFIED 0 zero counts
Filter


radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
CpuUtil
CpuAvail
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time Kind Proc Name Streams 25% Done 50% Done 75% Done 100% Done
5.294622 PAR_REGION_EXIT 12 radix_sort join 1 5.294501 5.294502 5.294505 5.294522
5.294623 PAR_REGION_EXIT 11 radix_sort join 1 5.294508 5.294510 5.294510 5.294513
5.294624 PAR_REGION_EXIT 6 radix_sort join 1 5.294504 5.294506 5.294507 5.294522
5.294626 PAR_REGION_EXIT 9 radix_sort join 1 5.294500 5.294513 5.294516 5.294523
5.294998 USER_SPECIFIED 0 find positions
5.295411 USER_SPECIFIED 0 shift elements
9.070591 USER_SPECIFIED 0 swap arrays
9.070640 USER_SPECIFIED 0 zero counts
Filter


radix.v1.r1.ap2
File
Help
Tview
Canal
Traced 538 events, dropped 0 in 36.24 seconds on 16 CPUs running at 500MHz, traps: 2182 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
CpuUtil
CpuAvail
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
Events
Time Kind Proc Name Streams 25% Done 50% Done 75% Done 100% Done
5.294622 PAR_REGION_EXIT 12 radix_sort join 1 5.294501 5.294502 5.294505 5.294522
5.294623 PAR_REGION_EXIT 11 radix_sort join 1 5.294508 5.294510 5.294510 5.294513
5.294624 PAR_REGION_EXIT 6 radix_sort join 1 5.294504 5.294506 5.294507 5.294522
5.294626 PAR_REGION_EXIT 9 radix_sort join 1 5.294500 5.294513 5.294516 5.294523
5.294998 USER_SPECIFIED 0 find positions
5.295411 USER_SPECIFIED 0 shift elements
9.070591 USER_SPECIFIED 0 swap arrays
9.070640 USER_SPECIFIED 0 zero counts
Filter


radix.v1.r1.ap2
File
Help
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
1 - #pragma mta trace "shift elements" 2048 2048 8
2 -
40 for (i = 0; i < size; ++i) { 40 16 9
8 SS j = pos[MTA_BIT_PACK(~mask, src[i])]++; 32000032 16000024 8000000
8 SS dst[j] = src[i]; 24000018 8000009 8000000
} 1
45 2 - #pragma mta trace "swap arrays" 8 8 8
2 - tmp = src; 24 8 8
2 S* src = dst; 16 16 8
2 S* dst = tmp; 9 8 8
50 2 S* mask <<= 8; 8 8 8
Loops
Details

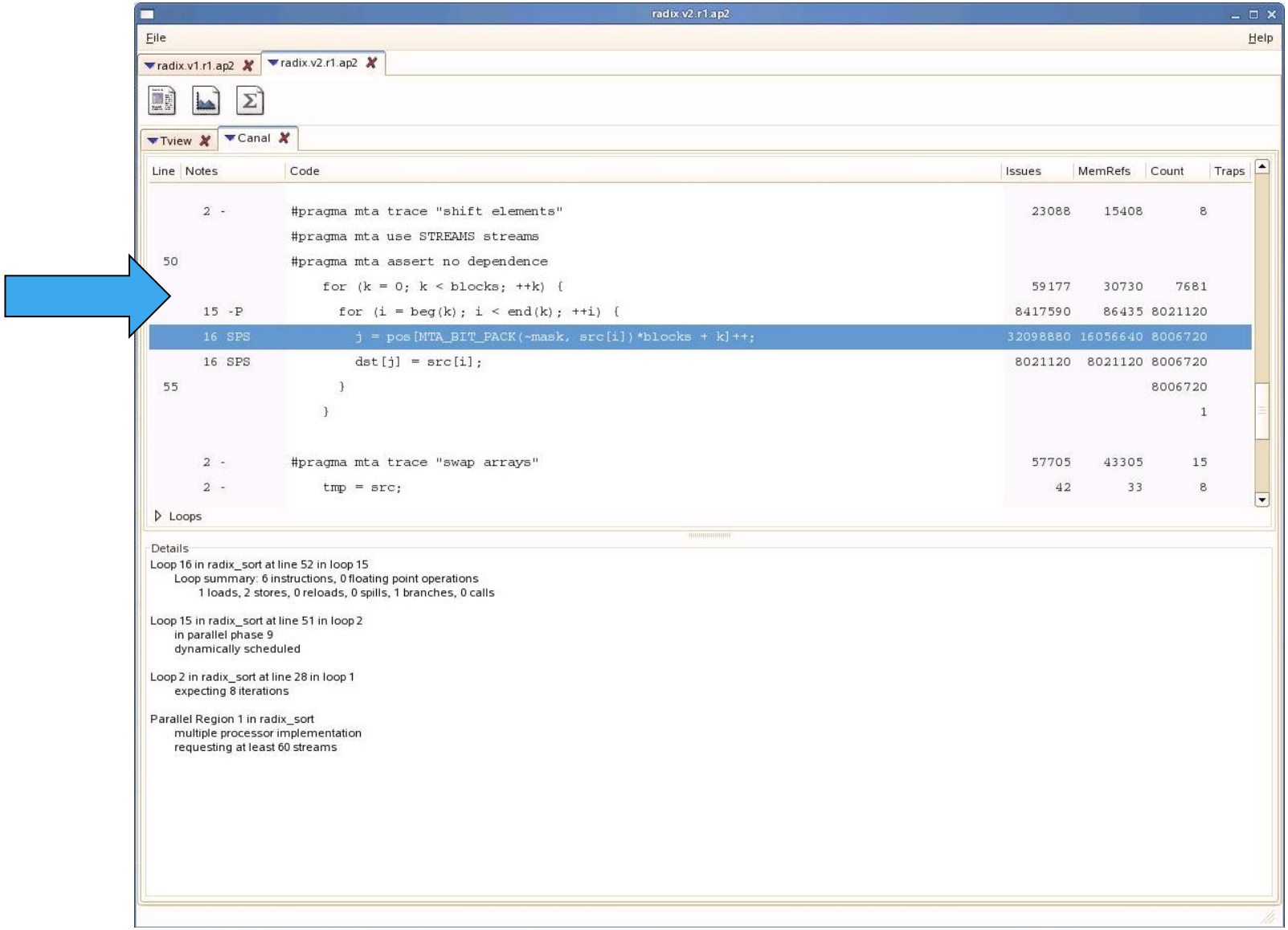
Workflow: Radix Sort III

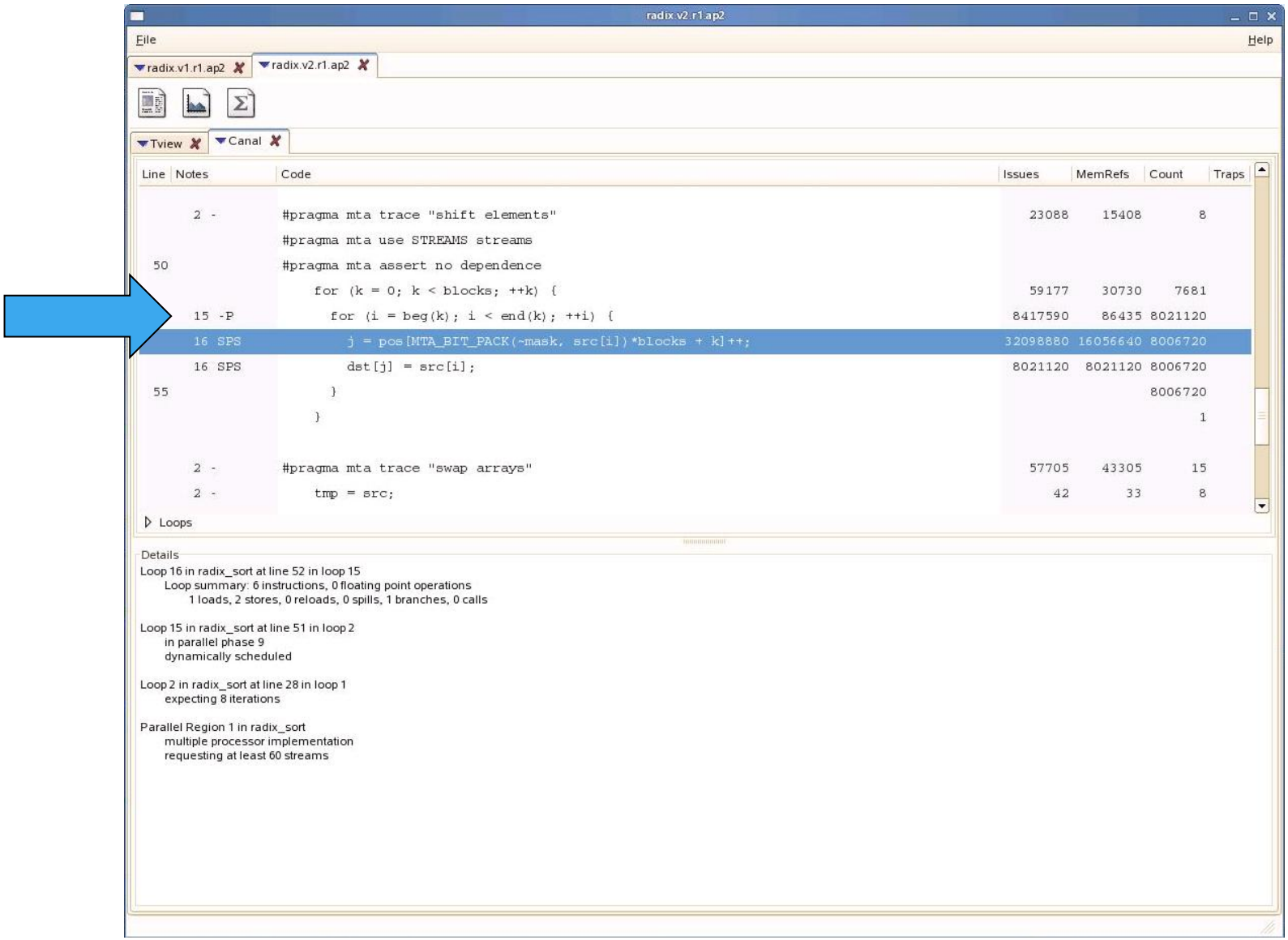
Further modify code

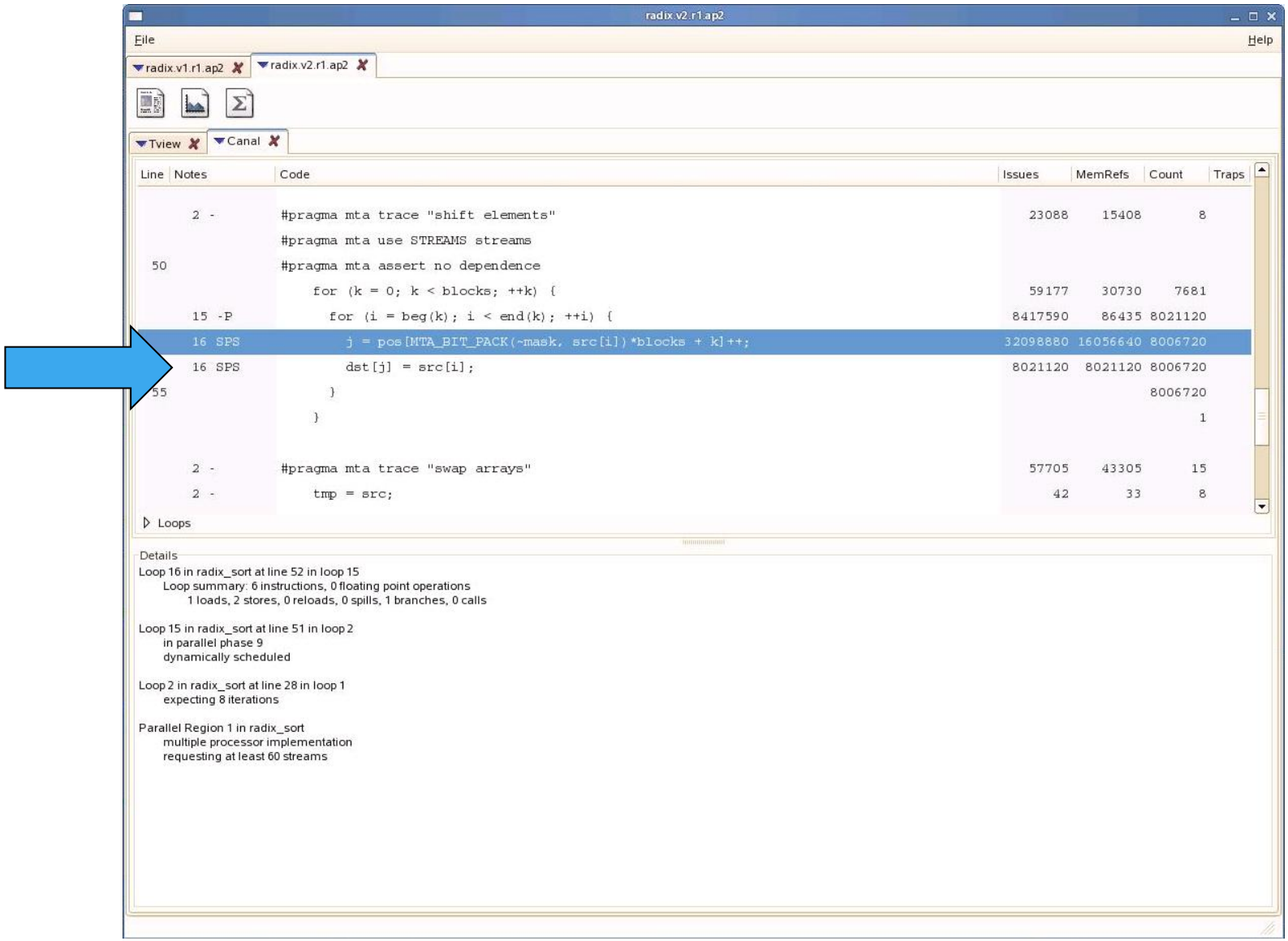
- Split the array up into sequential blocks, allowing us to replicate the buckets into which we determine the element counts and new positions.

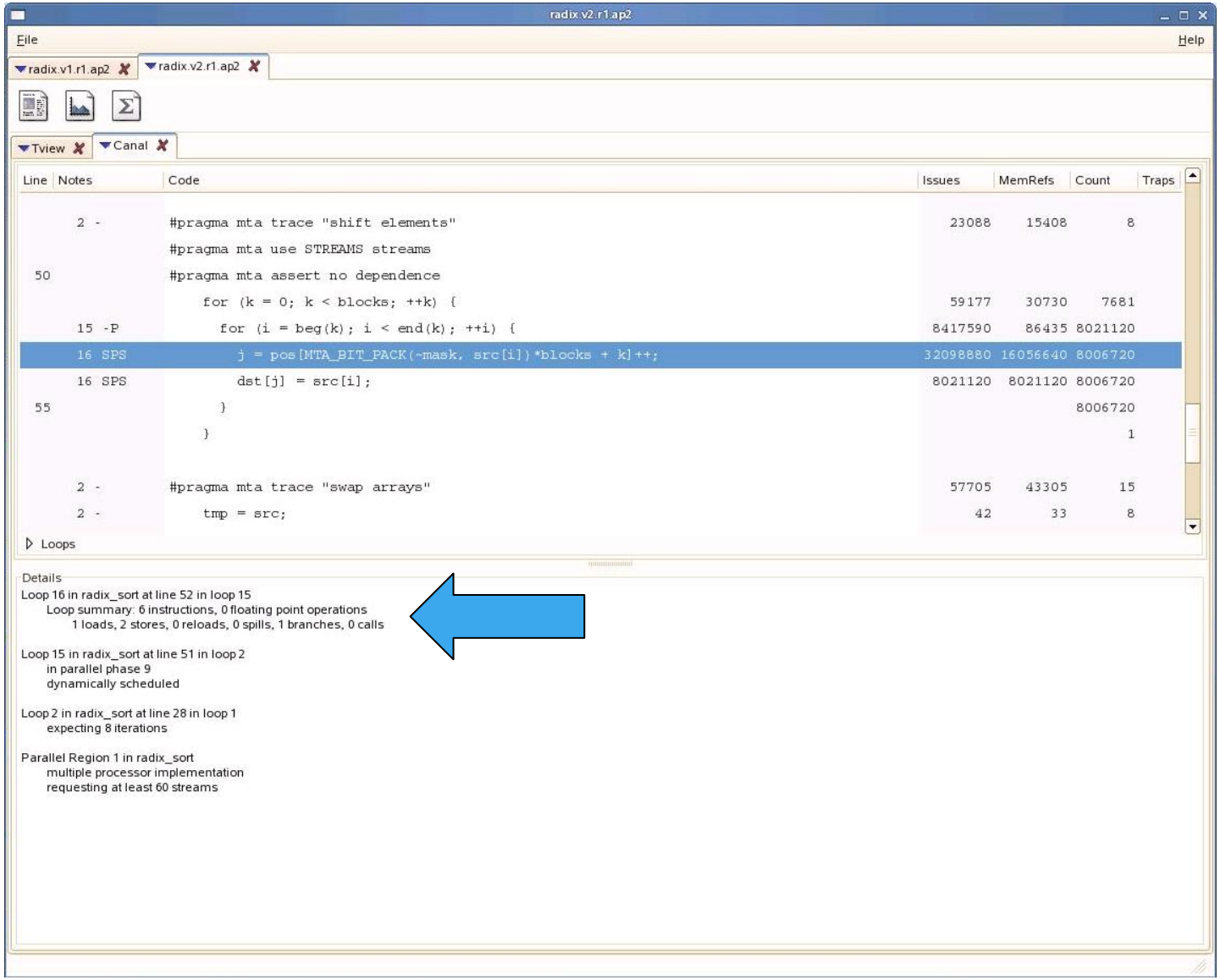
```
#pragma mta assert no dependence
    for (k = 0; k < blocks; ++k) {
      for (i = beg(k); i < end(k); ++i) {
        j = pos[MTA_BIT_PACK(~mask,
            src[i])*blocks + k]++;
        dst[j] = src[i];
      }
    }
```

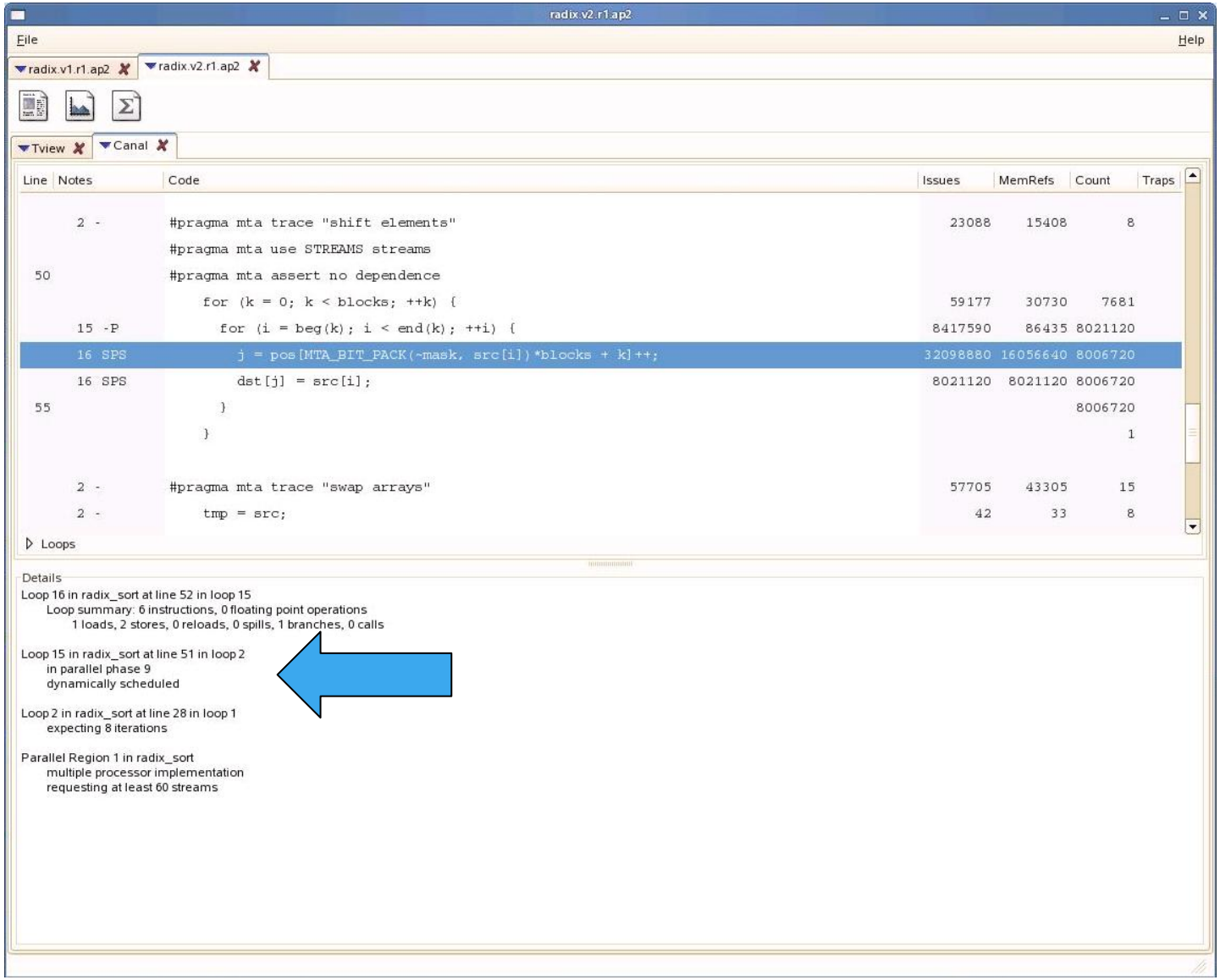
Recompile, run, post-process, and run Apprentice2

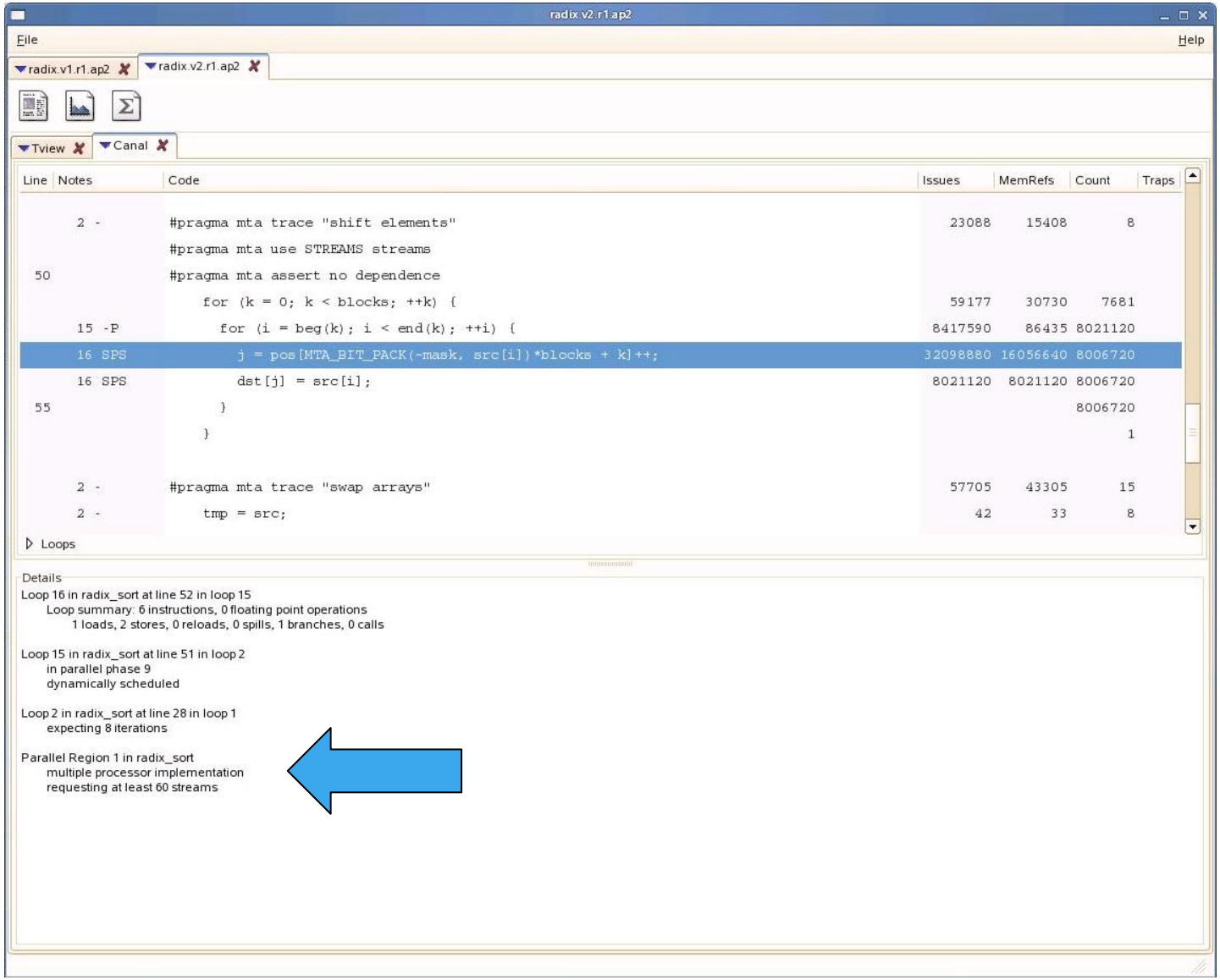

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

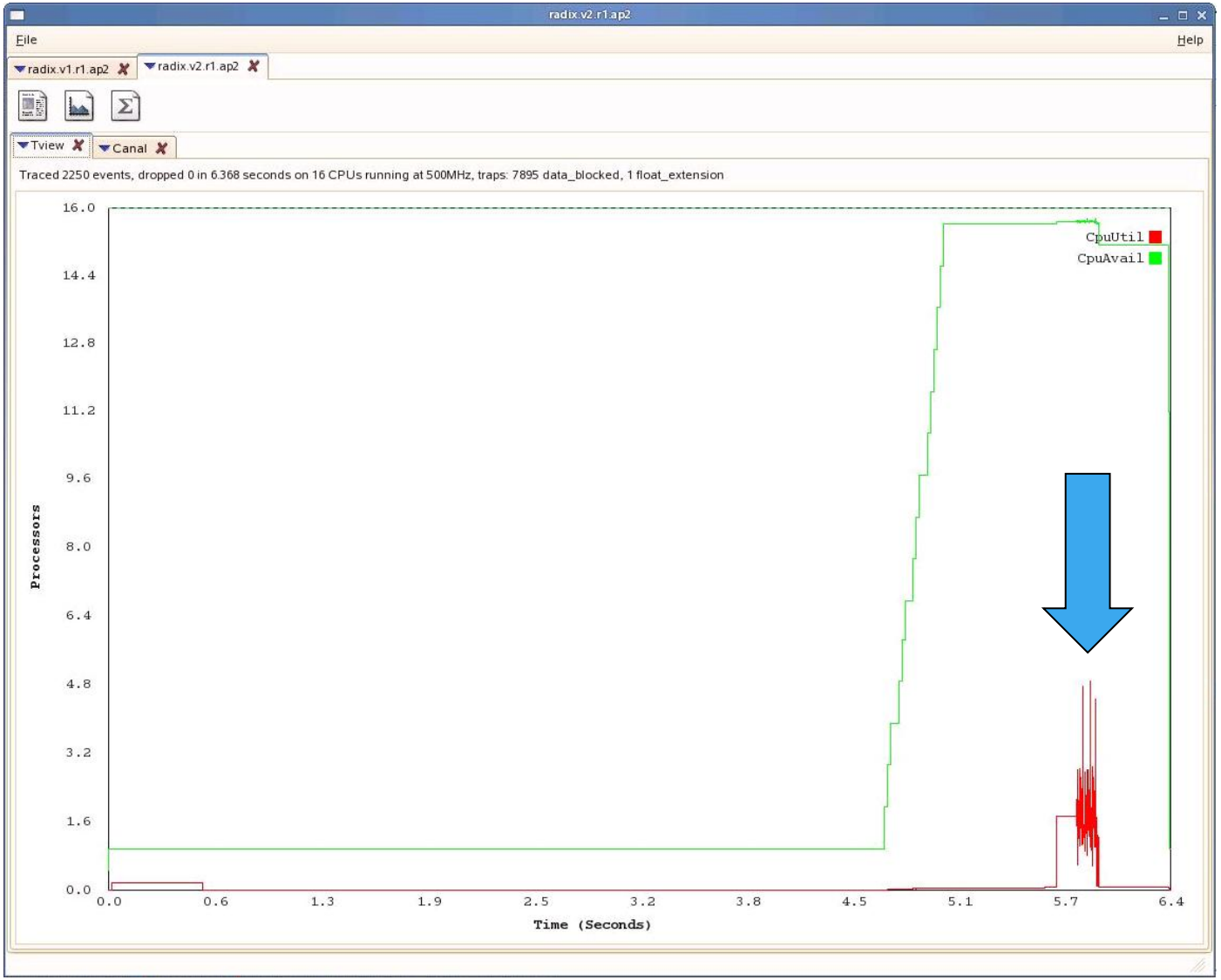

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams


radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

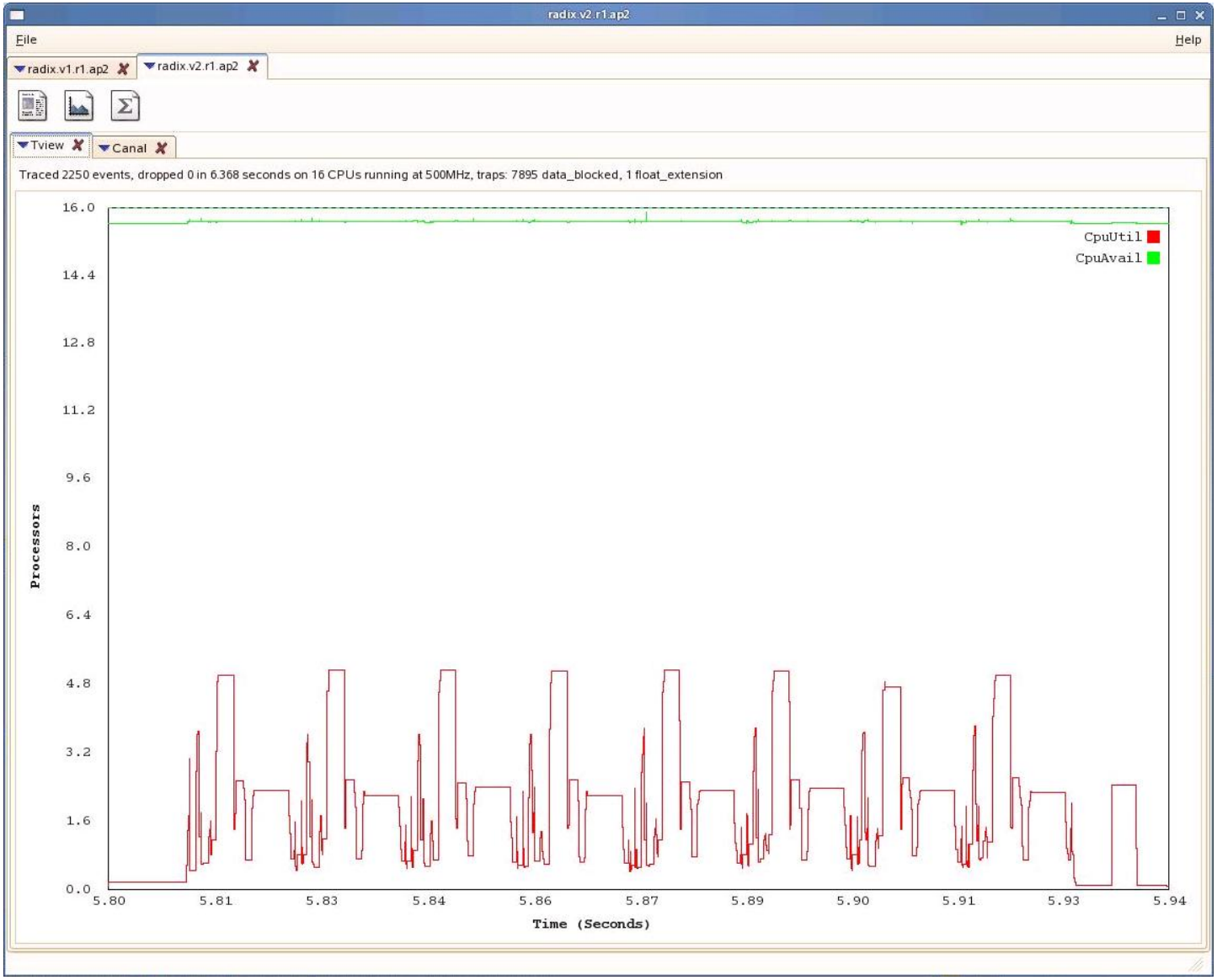

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

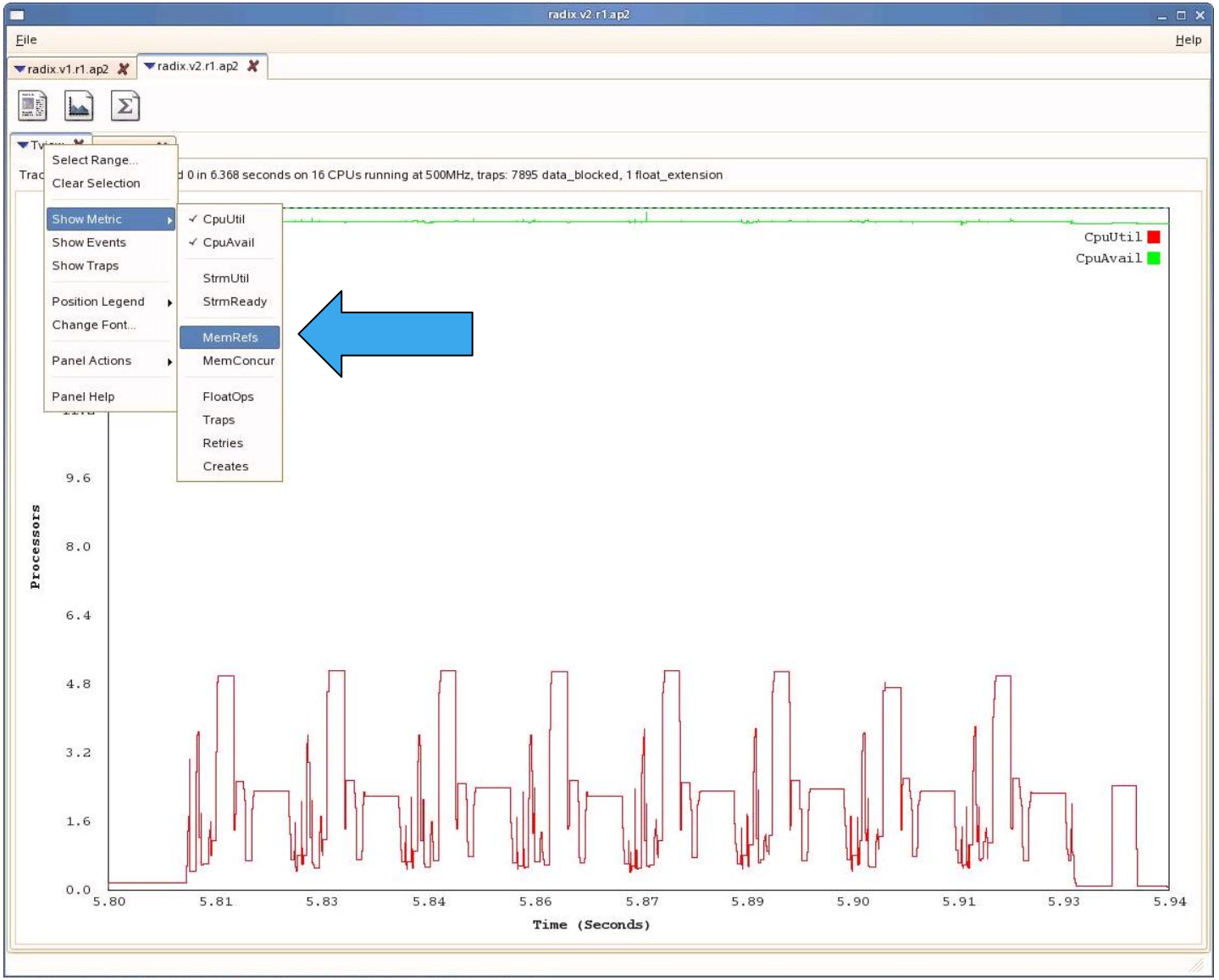

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

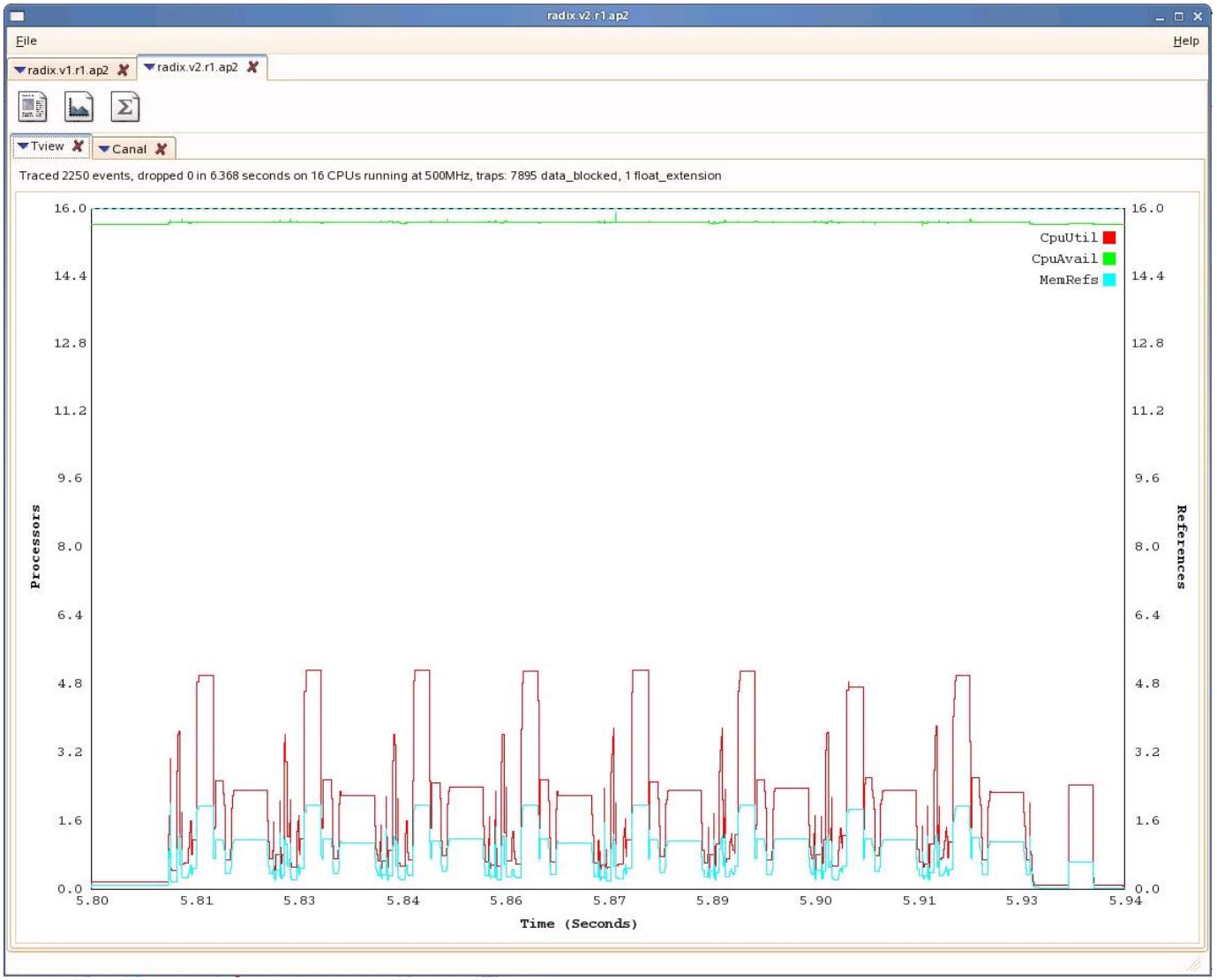

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

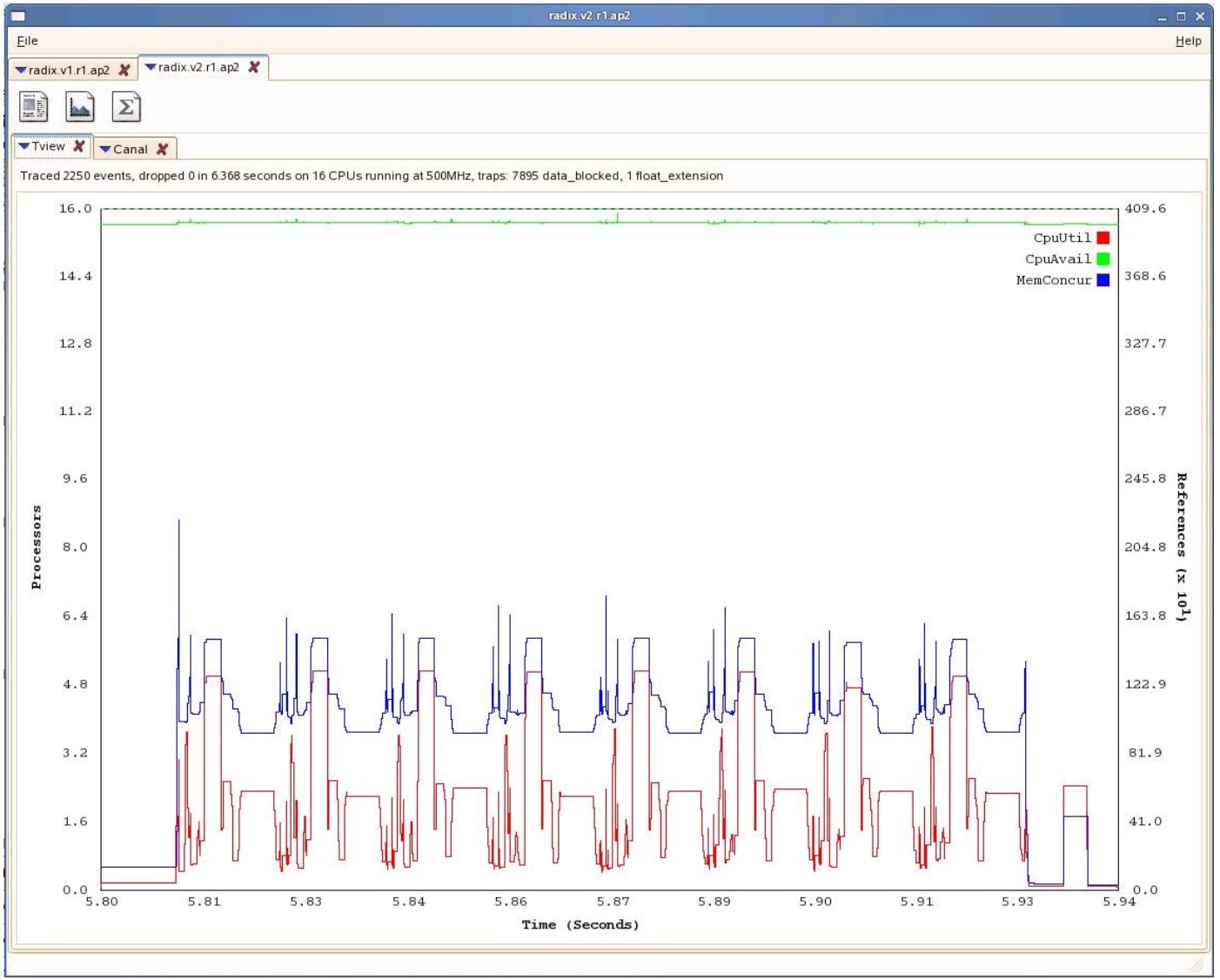

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
CpuUtil
CpuAvail
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
0.0
0.6
1.3
1.9
2.5
3.2
3.8
4.5
5.1
5.7
6.4
Time (Seconds)

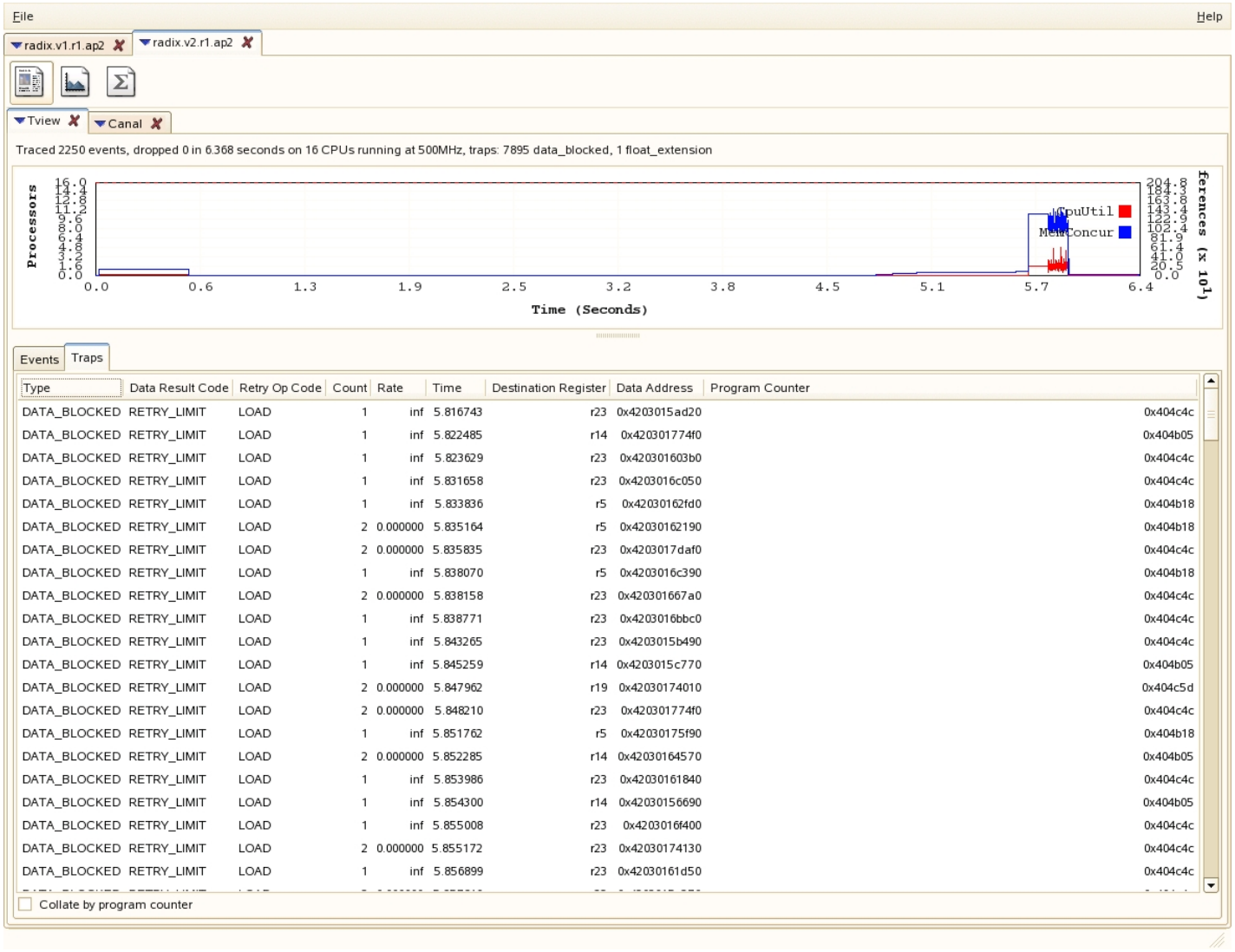

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
CpuUtil
CpuAvail
0.0
0.6
1.3
1.9
2.5
3.2
3.8
4.5
5.1
5.7
6.4
Time (Seconds)

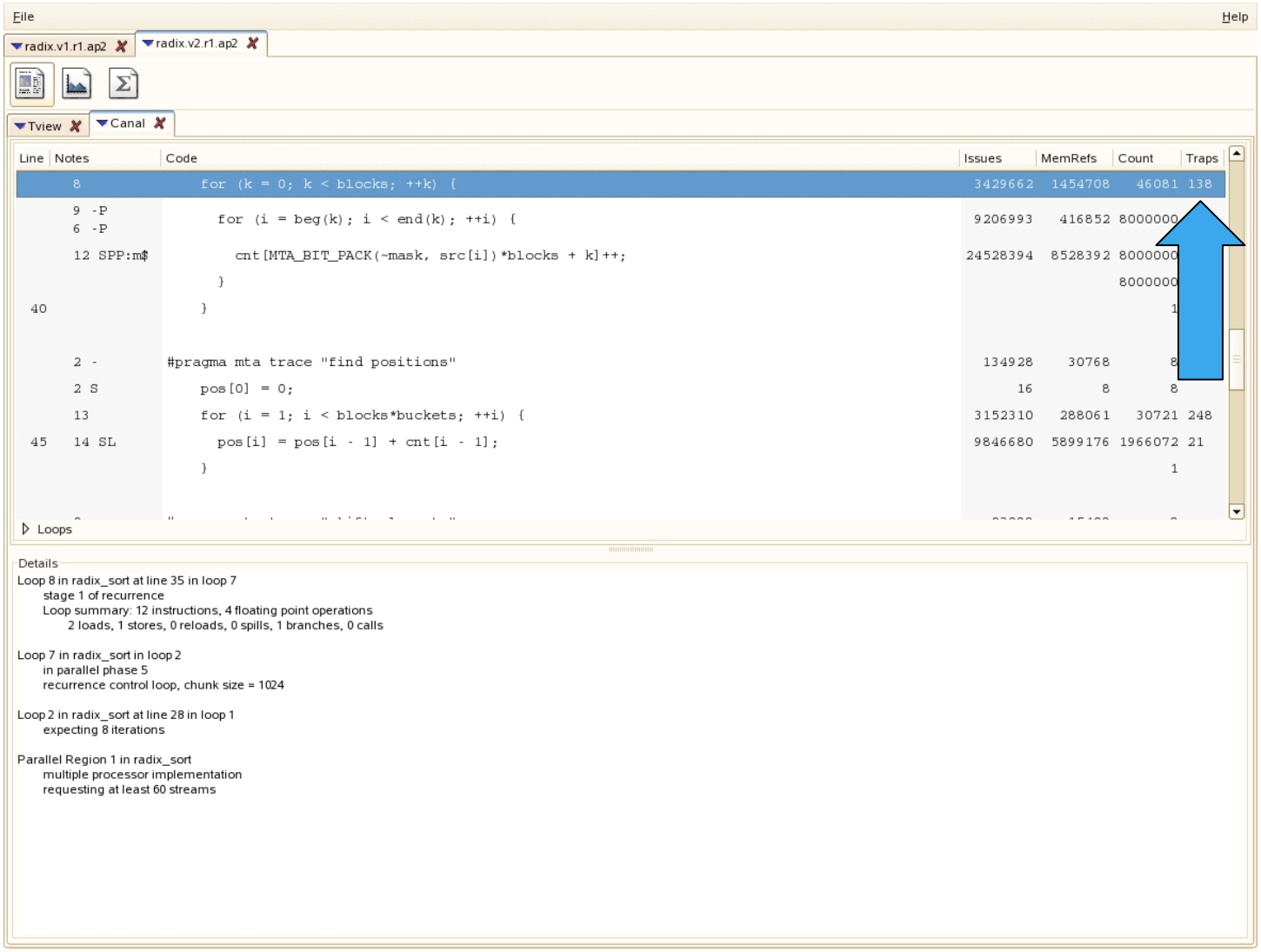

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
CpuUtil
CpuAvail
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
5.80
5.81
5.83
5.84
5.86
5.87
5.89
5.90
5.91
5.93
5.94
Time (Seconds)

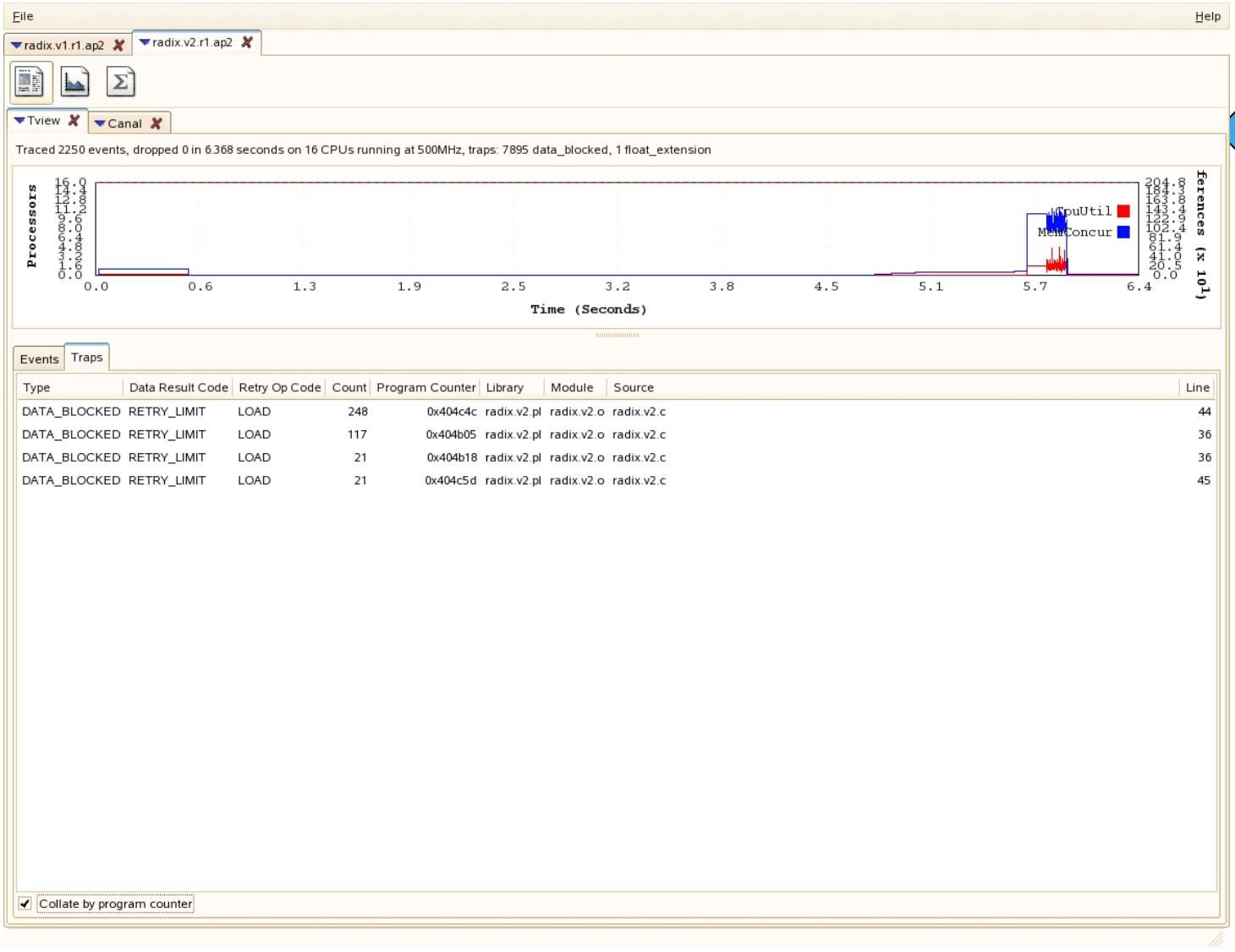

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Select Range...
Clear Selection
Show Metric
Show Events
Show Traps
Position Legend
Change Font...
Panel Actions
Panel Help
CpuUtil
CpuAvail
StrmUtil
StrmReady
MemRefs
MemConcur
FloatOps
Traps
Retries
Creates
0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
CpuUtil
CpuAvail
Processors
9.6
8.0
6.4
4.8
3.2
1.6
0.0
5.80
5.81
5.83
5.84
5.86
5.87
5.89
5.90
5.91
5.93
5.94
Time (Seconds)

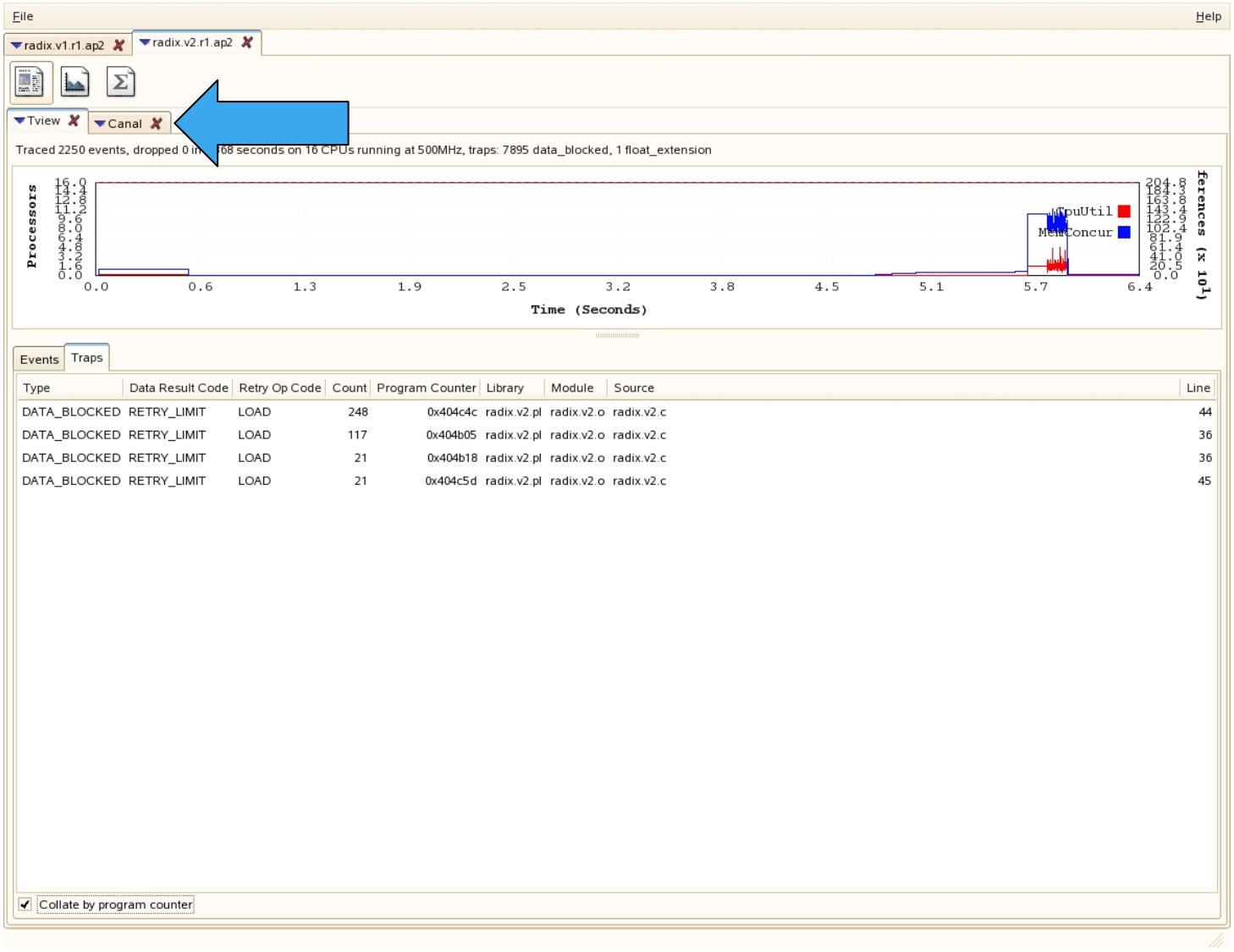

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemRefs
Processors
References
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
5.80
5.81
5.83
5.84
5.86
5.87
5.89
5.90
5.91
5.93
5.94
Time (Seconds)

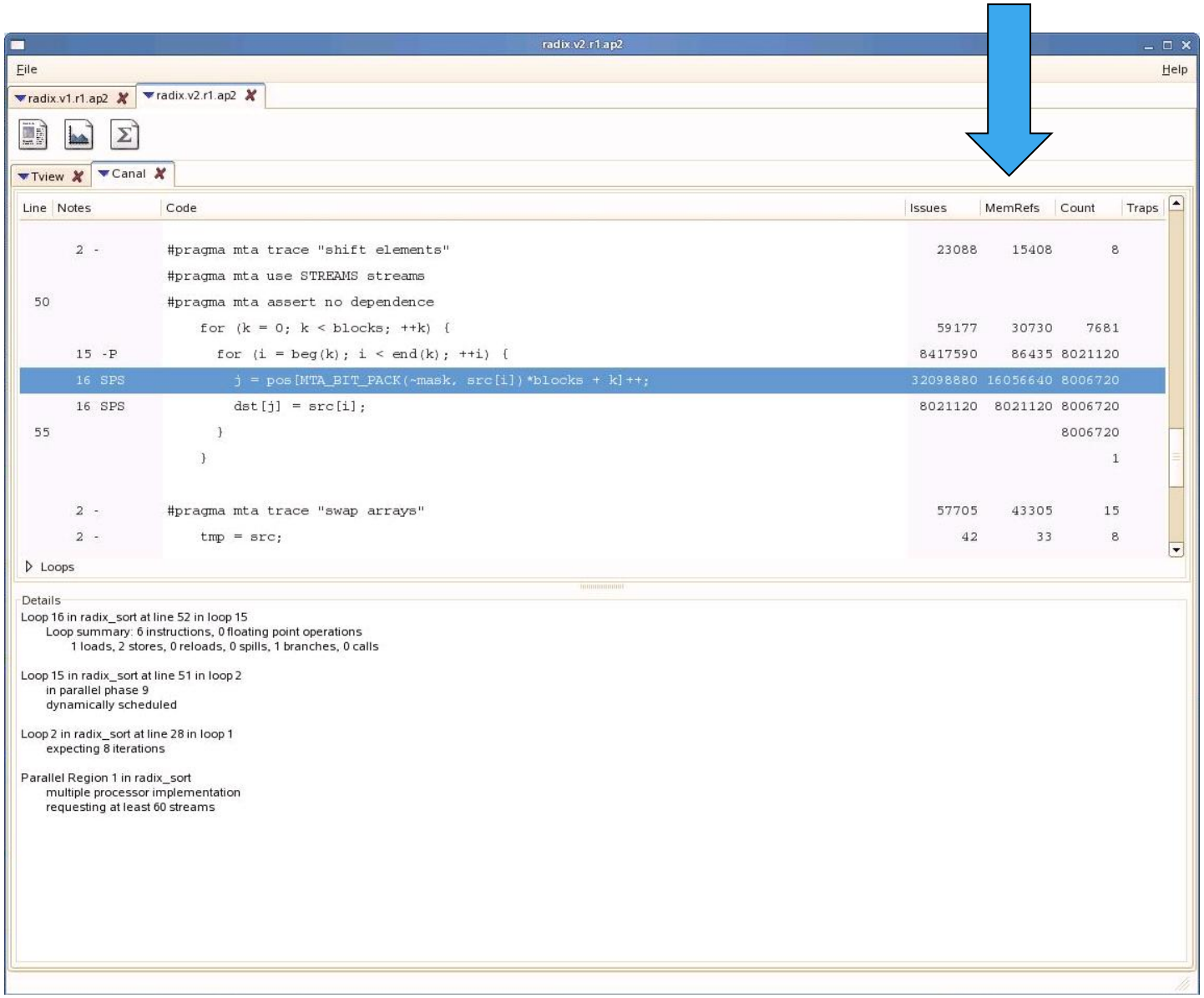

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
CpuUtil
CpuAvail
MemConcur
Processors
References (x 10^1)
Time (Seconds)
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
409.6
368.6
327.7
286.7
245.8
204.8
163.8
122.9
81.9
41.0
0.0
5.80
5.81
5.83
5.84
5.86
5.87
5.89
5.90
5.91
5.93
5.94

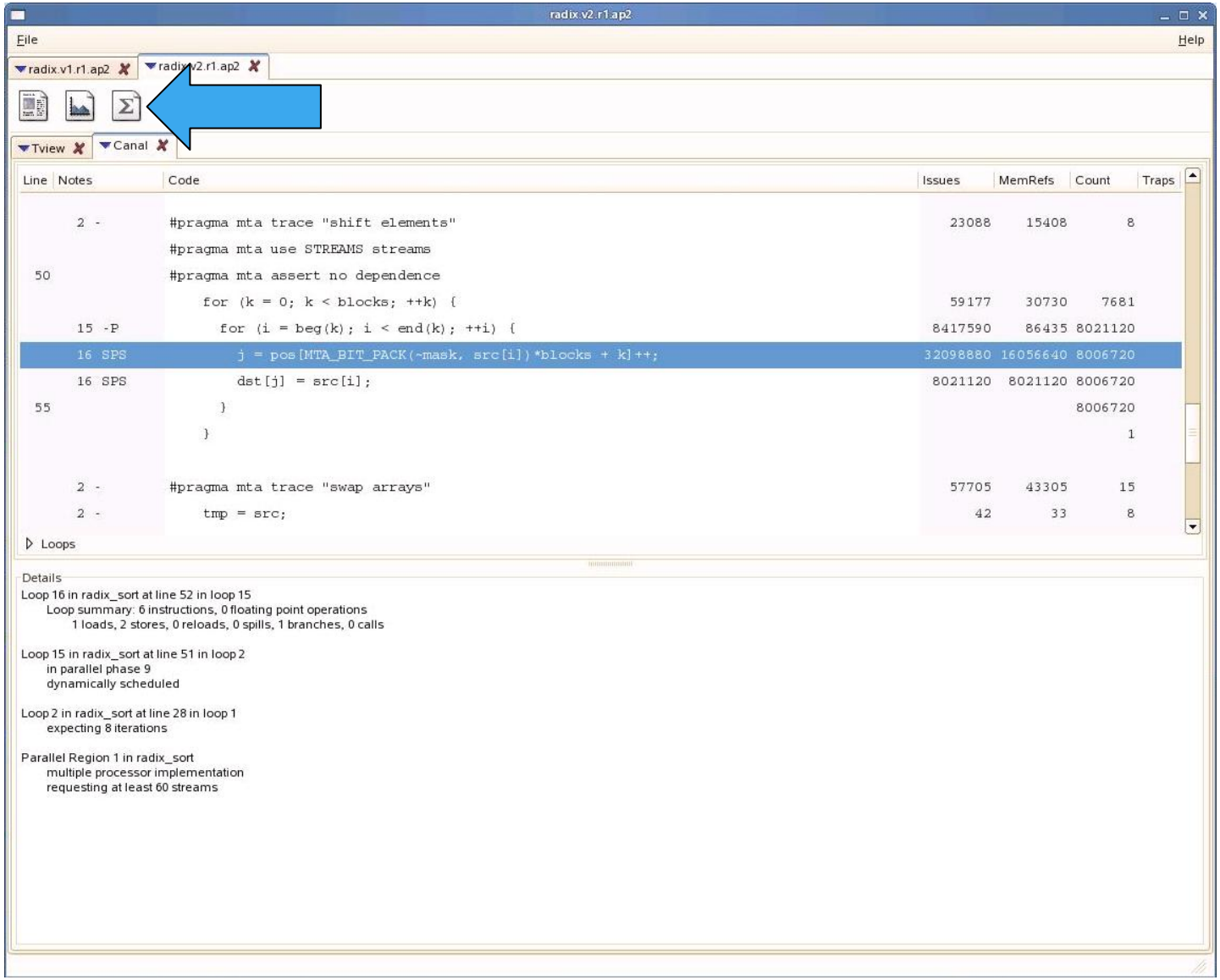

File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
0.0
0.6
1.3
1.9
2.5
3.2
3.8
4.5
5.1
5.7
6.4
Time (Seconds)
CpuUtil
MemConcur
204.8
184.3
163.8
143.4
122.9
102.4
81.9
61.4
41.0
20.5
0.0
Events
Traps
Type
Data Result Code
Retry Op Code
Count
Rate
Time
Destination Register
Data Address
Program Counter
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.816743 r23 0x4203015ad20 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.822485 r14 0x420301774f0 0x404b05
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.823629 r23 0x420301603b0 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.831658 r23 0x4203016c050 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.833836 r5 0x42030162fd0 0x404b18
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.835164 r5 0x42030162190 0x404b18
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.835835 r23 0x4203017daf0 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.838070 r5 0x4203016c390 0x404b18
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.838158 r23 0x420301667a0 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.838771 r23 0x4203016bbc0 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.843265 r23 0x4203015b490 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.845259 r14 0x4203015c770 0x404b05
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.847962 r19 0x42030174010 0x404c5d
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.848210 r23 0x420301774f0 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.851762 r5 0x42030175f90 0x404b18
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.852285 r14 0x42030164570 0x404b05
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.853986 r23 0x42030161840 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.854300 r14 0x42030156690 0x404b05
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.855008 r23 0x4203016f400 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 2 0.000000 5.855172 r23 0x42030174130 0x404c4c
DATA_BLOCKED RETRY_LIMIT LOAD 1 inf 5.856899 r23 0x42030161d50 0x404c4c
Collate by program counter

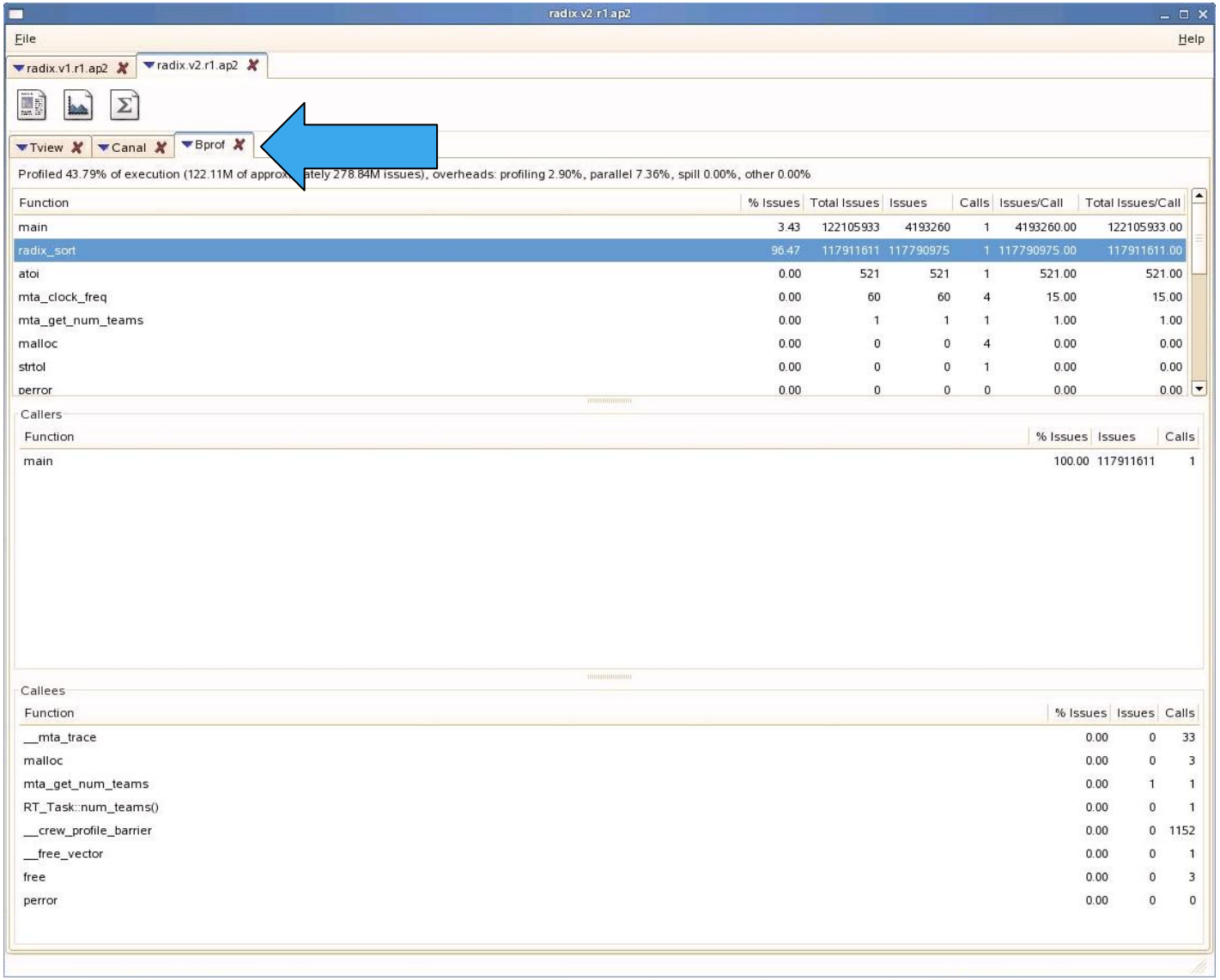

File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
8 for (k = 0; k < blocks; ++k) { 3429662 1454708 46081 138
9 -P for (i = beg(k); i < end(k); ++i) { 9206993 416852 8000000
6 -P
12 SPP:m$ cnt[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 24528394 8528392 8000000
} 8000000
40 } 1
2 - #pragma mta trace "find positions" 134928 30768 8
2 S pos[0] = 0; 16 8 8
13 for (i = 1; i < blocks*buckets; ++i) { 3152310 288061 30721 248
45 14 SL pos[i] = pos[i - 1] + cnt[i - 1]; 9846680 5899176 1966072 21
} 1
Loops
Details
Loop 8 in radix_sort at line 35 in loop 7
stage 1 of recurrence
Loop summary: 12 instructions, 4 floating point operations
2 loads, 1 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 7 in radix_sort in loop 2
in parallel phase 5
recurrence control loop, chunk size = 1024
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

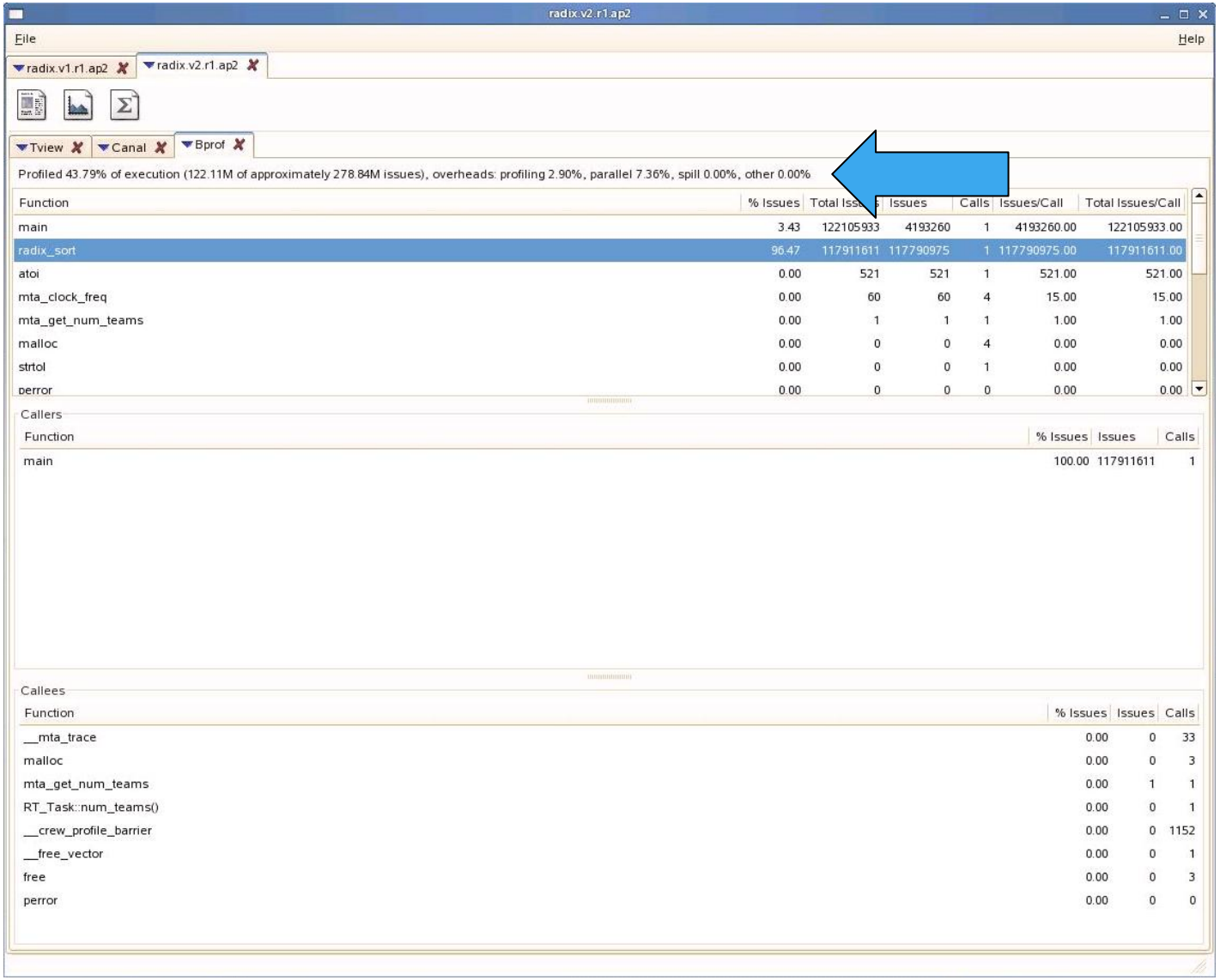

File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in 6.368 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
0.0
0.6
1.3
1.9
2.5
3.2
3.8
4.5
5.1
5.7
6.4
Time (Seconds)
CpuUtil
MemConcur
204.8
184.3
163.8
143.4
122.9
102.4
81.9
61.4
41.0
20.5
0.0
ferences (x 10^1)
Events
Traps
Type
Data Result Code
Retry Op Code
Count
Program Counter
Library
Module
Source
Line
DATA_BLOCKED RETRY_LIMIT LOAD 248 0x404c4c radix.v2.pl radix.v2.o radix.v2.c 44
DATA_BLOCKED RETRY_LIMIT LOAD 117 0x404b05 radix.v2.pl radix.v2.o radix.v2.c 36
DATA_BLOCKED RETRY_LIMIT LOAD 21 0x404b18 radix.v2.pl radix.v2.o radix.v2.c 36
DATA_BLOCKED RETRY_LIMIT LOAD 21 0x404c5d radix.v2.pl radix.v2.o radix.v2.c 45
Collate by program counter

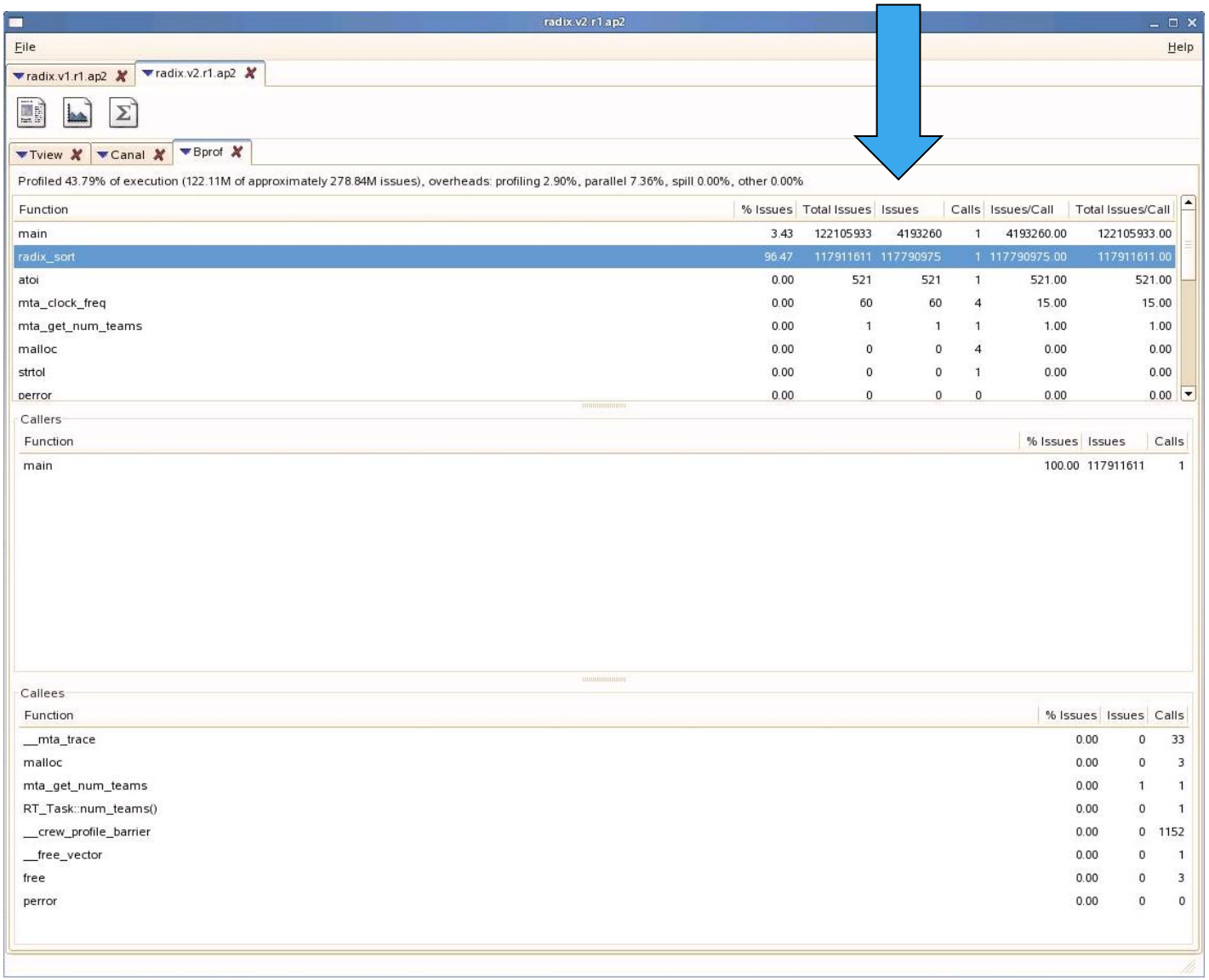

File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Traced 2250 events, dropped 0 in
68 seconds on 16 CPUs running at 500MHz, traps: 7895 data_blocked, 1 float_extension
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
0.0
0.6
1.3
1.9
2.5
3.2
3.8
4.5
5.1
5.7
6.4
Time (Seconds)
CpuUtil
MemConcur
204.8
184.3
163.8
143.4
122.9
102.4
81.9
61.4
41.0
20.5
0.0
ferences (x 10^1)
Events
Traps
Type
Data Result Code
Retry Op Code
Count
Program Counter
Library
Module
Source
Line
DATA_BLOCKED RETRY_LIMIT LOAD 248 0x404c4c radix.v2.pl radix.v2.o radix.v2.c 44
DATA_BLOCKED RETRY_LIMIT LOAD 117 0x404b05 radix.v2.pl radix.v2.o radix.v2.c 36
DATA_BLOCKED RETRY_LIMIT LOAD 21 0x404b18 radix.v2.pl radix.v2.o radix.v2.c 36
DATA_BLOCKED RETRY_LIMIT LOAD 21 0x404c5d radix.v2.pl radix.v2.o radix.v2.c 45
Collate by program counter

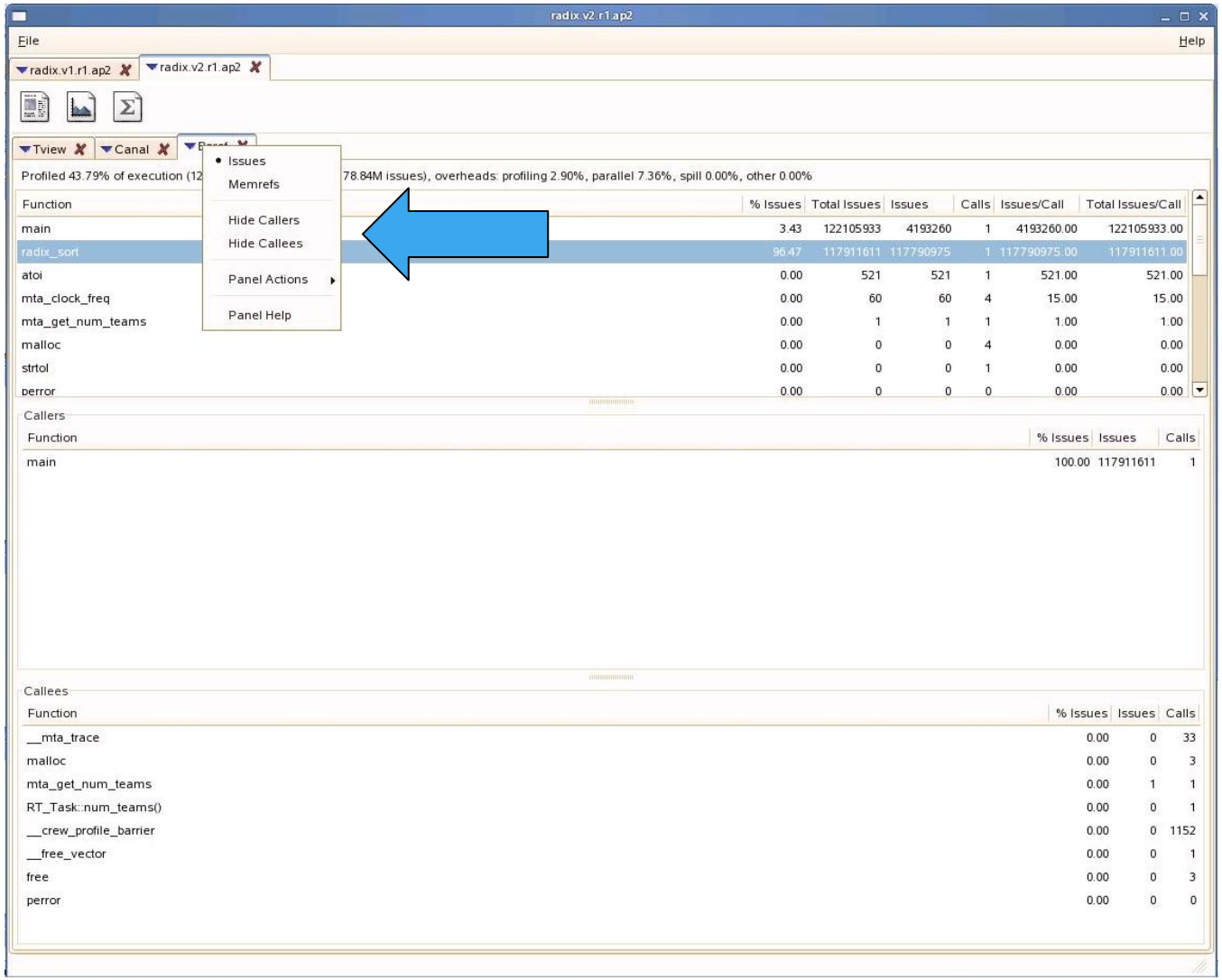

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

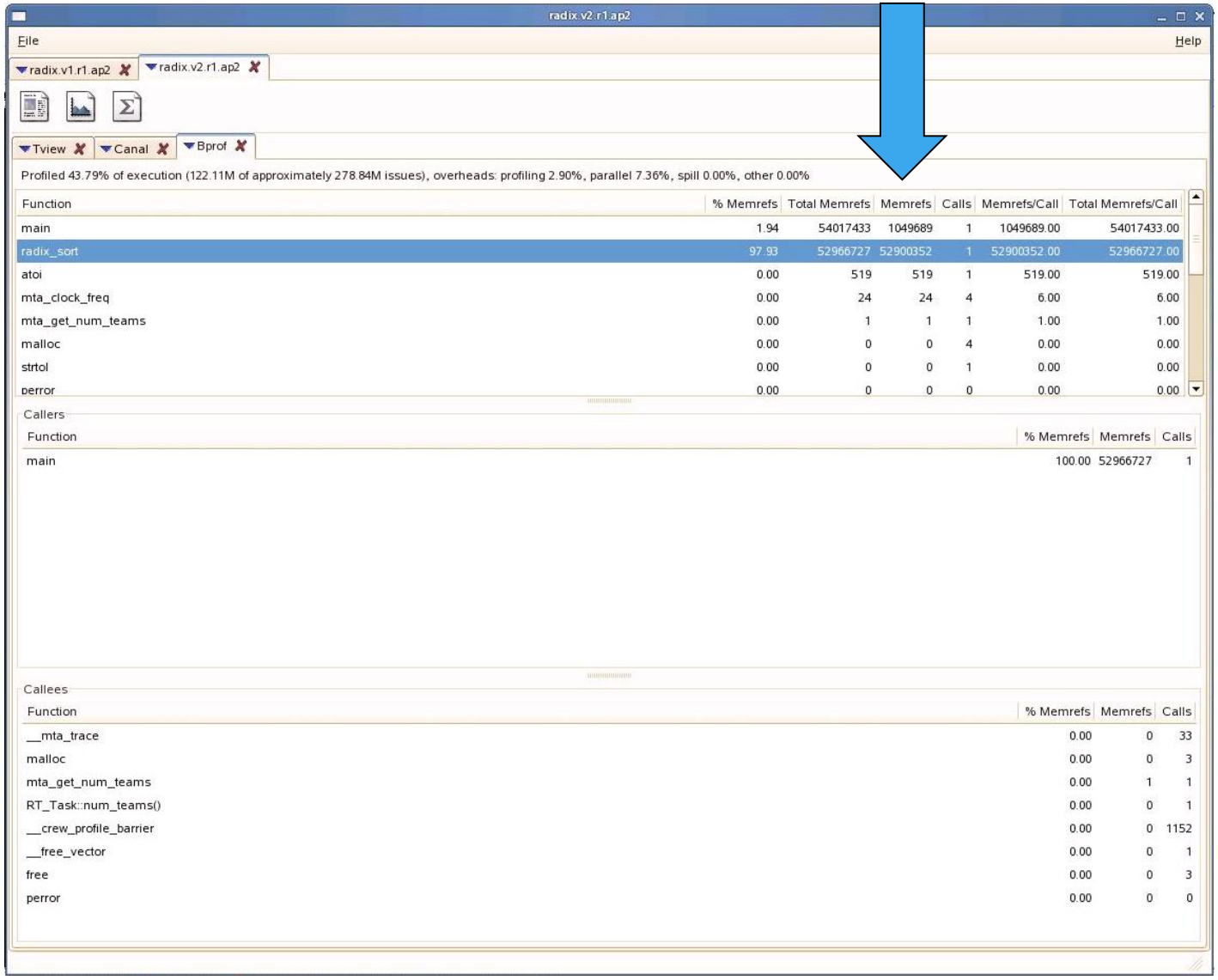

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Line Notes Code Issues MemRefs Count Traps
2 - #pragma mta trace "shift elements" 23088 15408 8
#pragma mta use STREAMS streams
50 #pragma mta assert no dependence
for (k = 0; k < blocks; ++k) { 59177 30730 7681
15 -P for (i = beg(k); i < end(k); ++i) { 8417590 86435 8021120
16 SPS j = pos[MTA_BIT_PACK(~mask, src[i])*blocks + k]++; 32098880 16056640 8006720
16 SPS dst[j] = src[i]; 8021120 8021120 8006720
55 } 8006720
} 1
2 - #pragma mta trace "swap arrays" 57705 43305 15
2 - tmp = src; 42 33 8
Loops
Details
Loop 16 in radix_sort at line 52 in loop 15
Loop summary: 6 instructions, 0 floating point operations
1 loads, 2 stores, 0 reloads, 0 spills, 1 branches, 0 calls
Loop 15 in radix_sort at line 51 in loop 2
in parallel phase 9
dynamically scheduled
Loop 2 in radix_sort at line 28 in loop 1
expecting 8 iterations
Parallel Region 1 in radix_sort
multiple processor implementation
requesting at least 60 streams

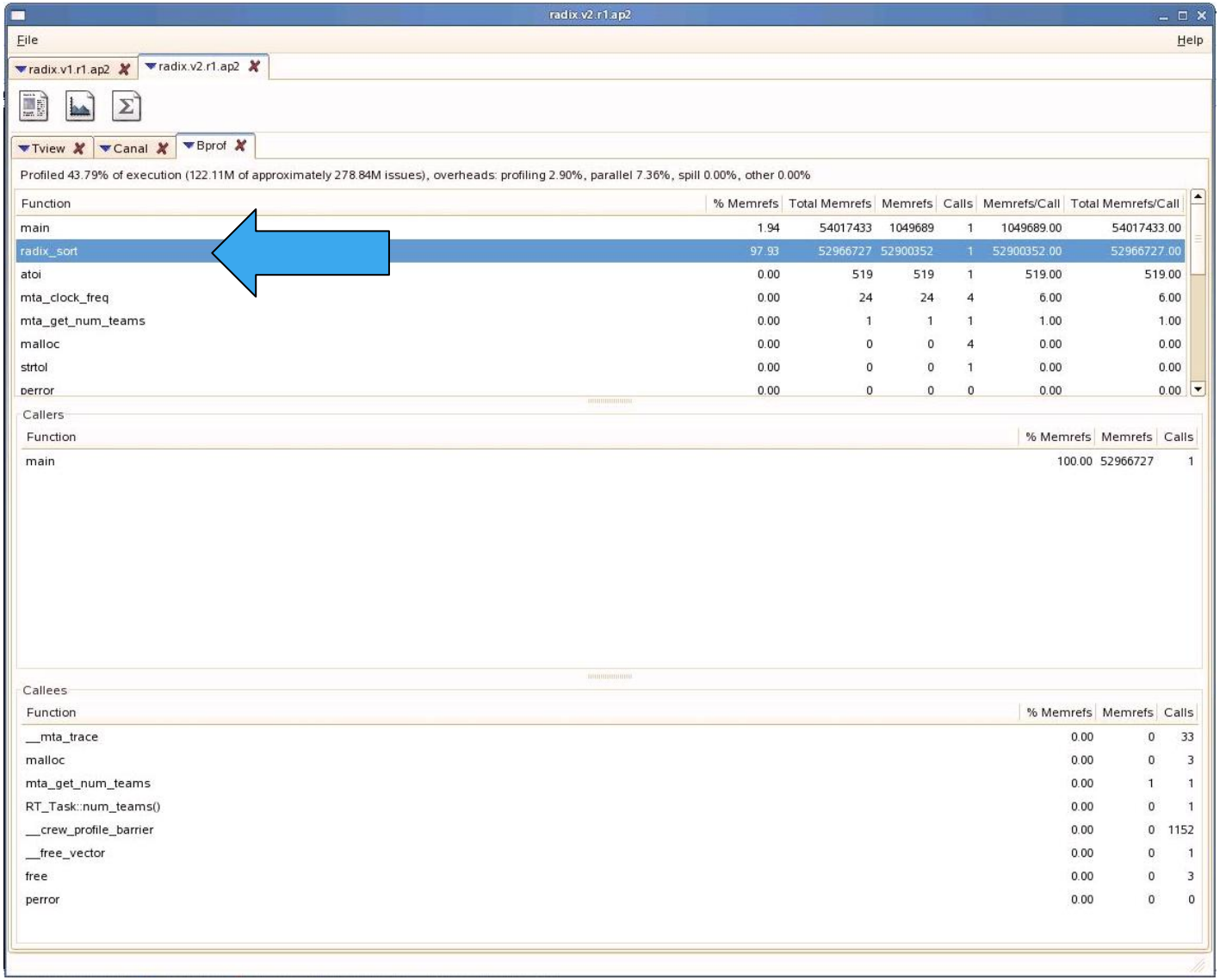

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Issues Total Issues Issues Calls Issues/Call Total Issues/Call
main 3.43 122105933 4193260 1 4193260.00 122105933.00
radix_sort 96.47 117911611 117790975 1 117790975.00 117911611.00
atoi 0.00 521 521 1 521.00 521.00
mta_clock_freq 0.00 60 60 4 15.00 15.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Issues Issues Calls
main 100.00 117911611 1
Callees
Function % Issues Issues Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

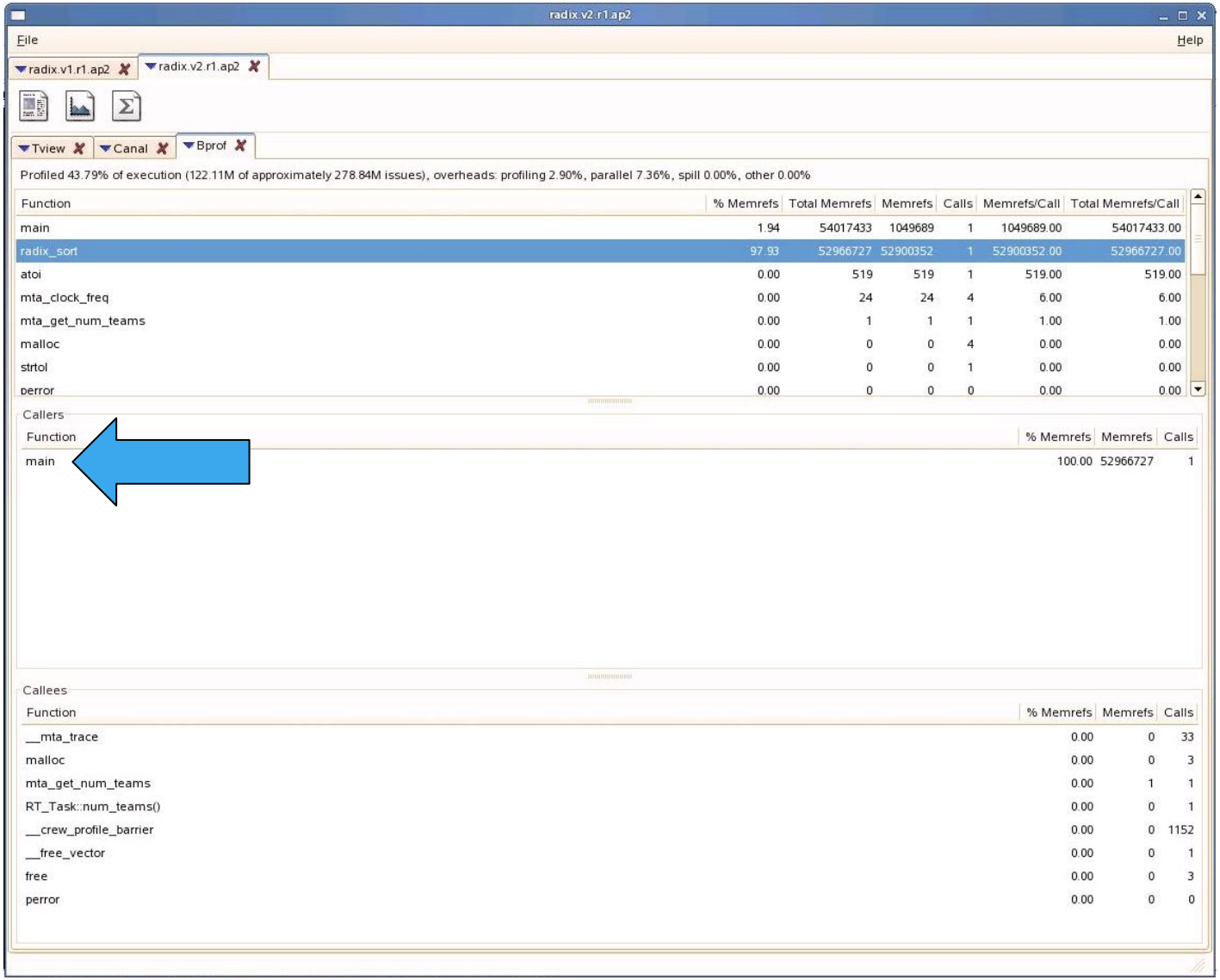

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Issues Total Issues Issues Calls Issues/Call Total Issues/Call
main 3.43 122105933 4193260 1 4193260.00 122105933.00
radix_sort 96.47 117911611 117790975 1 117790975.00 117911611.00
atoi 0.00 521 521 1 521.00 521.00
mta_clock_freq 0.00 60 60 4 15.00 15.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Issues Issues Calls
main 100.00 117911611 1
Callees
Function % Issues Issues Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

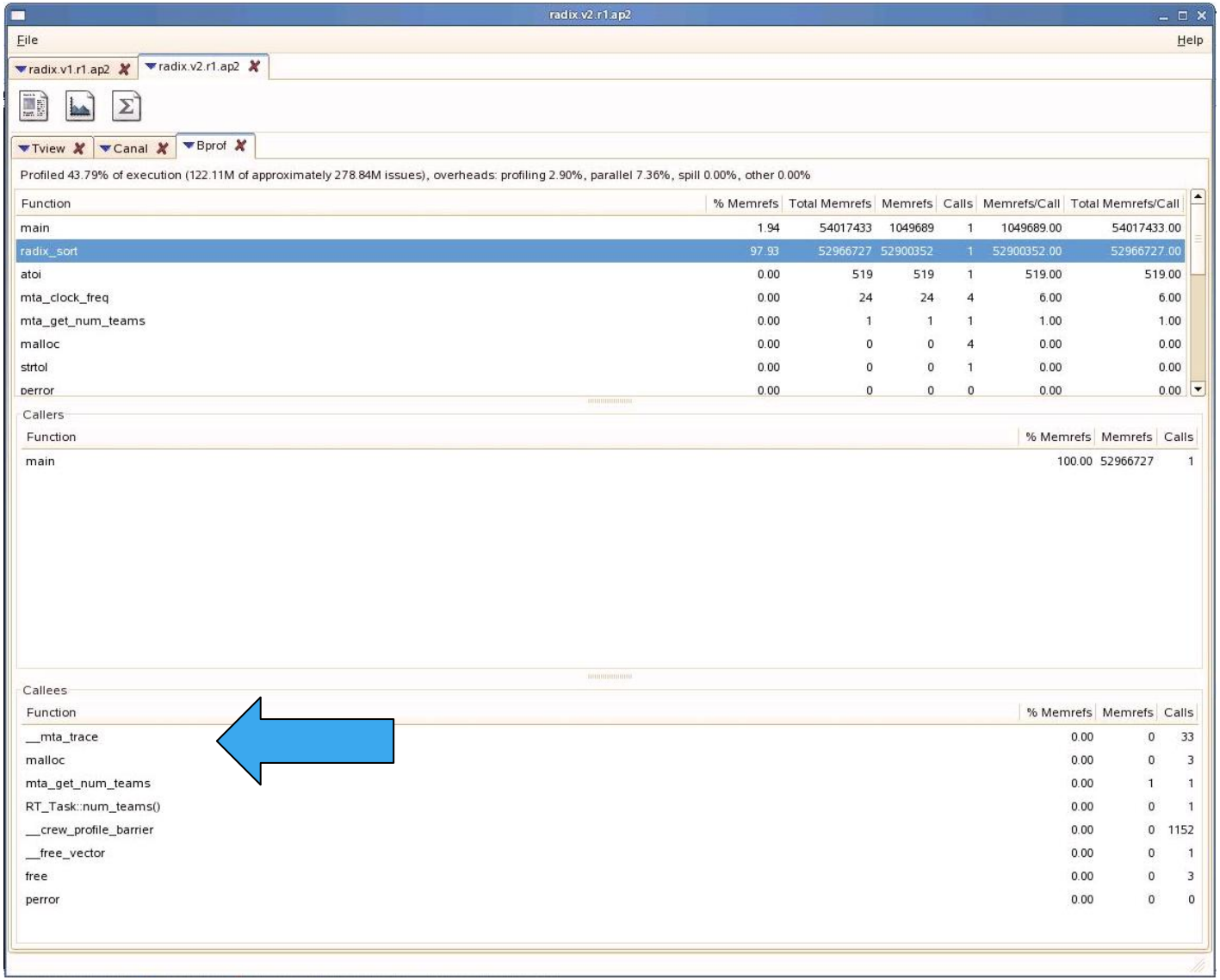

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Issues Total Issues Issues Calls Issues/Call Total Issues/Call
main 3.43 122105933 4193260 1 4193260.00 122105933.00
radix_sort 96.47 117911611 117790975 1 117790975.00 117911611.00
atoi 0.00 521 521 1 521.00 521.00
mta_clock_freq 0.00 60 60 4 15.00 15.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Issues Issues Calls
main 100.00 117911611 1
Callees
Function % Issues Issues Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

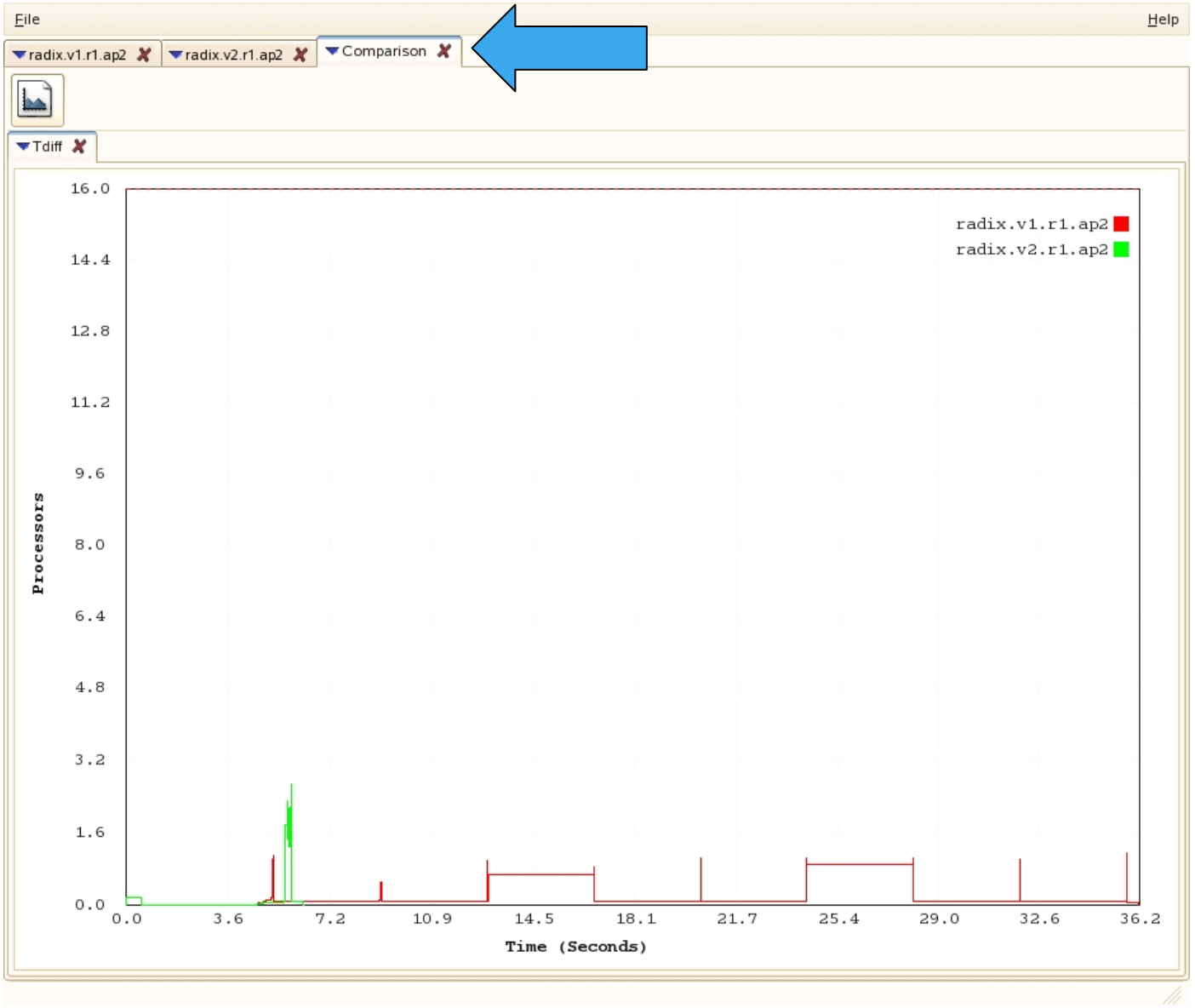

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Issues
Memrefs
Hide Callers
Hide Callees
Panel Actions
Panel Help
Profiled 43.79% of execution (12
78.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Issues Total Issues Issues Calls Issues/Call Total Issues/Call
main 3.43 122105933 4193260 1 4193260.00 122105933.00
radix_sort 96.47 117911611 117790975 1 117790975.00 117911611.00
atoi 0.00 521 521 1 521.00 521.00
mta_clock_freq 0.00 60 60 4 15.00 15.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Issues Issues Calls
main 100.00 117911611 1
Callees
Function % Issues Issues Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

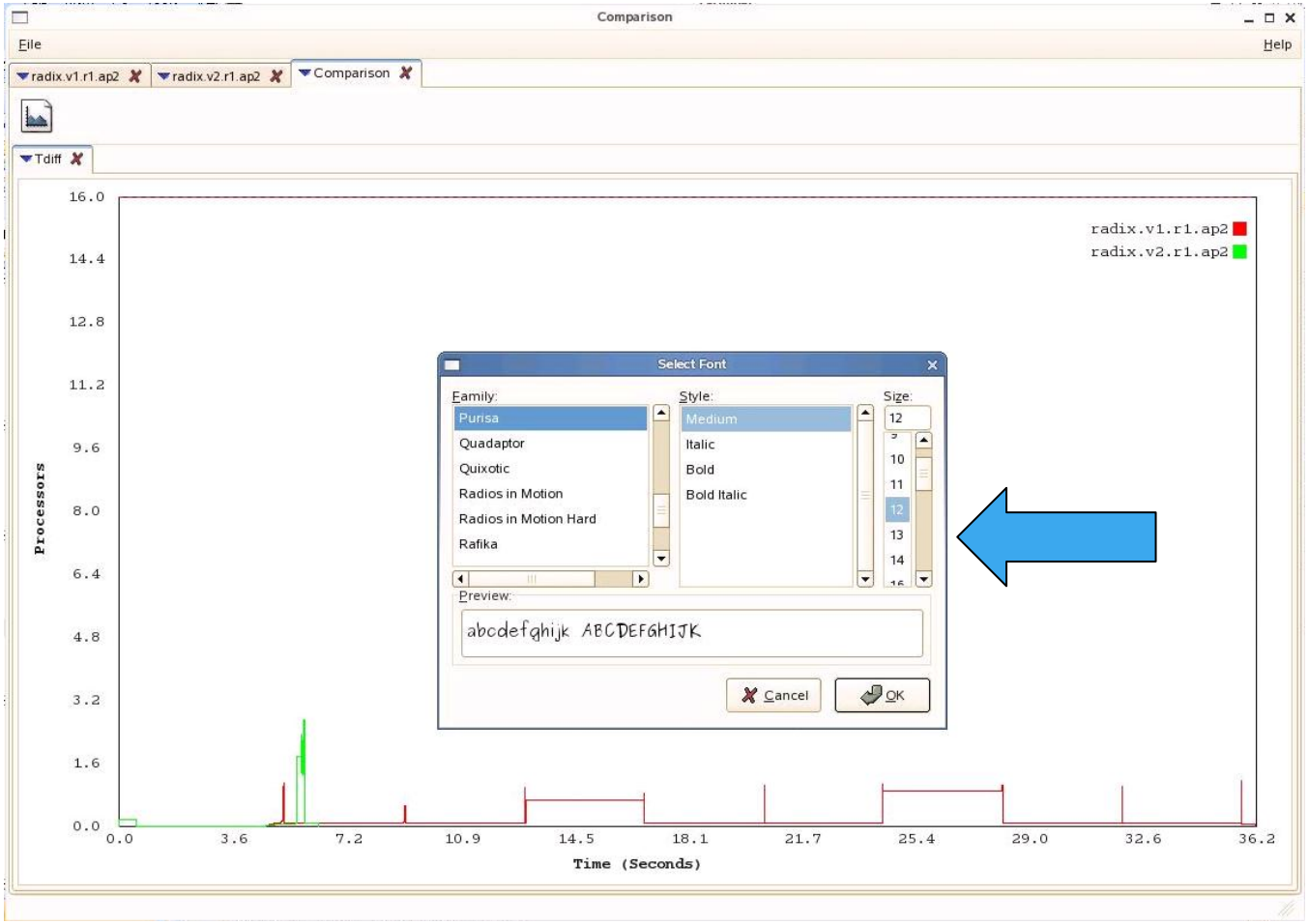

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Memrefs Total Memrefs Memrefs Calls Memrefs/Call Total Memrefs/Call
main 1.94 54017433 1049689 1 1049689.00 54017433.00
radix_sort 97.93 52966727 52900352 1 52900352.00 52966727.00
atoi 0.00 519 519 1 519.00 519.00
mta_clock_freq 0.00 24 24 4 6.00 6.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Memrefs Memrefs Calls
main 100.00 52966727 1
Callees
Function % Memrefs Memrefs Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0


radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Memrefs Total Memrefs Memrefs Calls Memrefs/Call Total Memrefs/Call
main 1.94 54017433 1049689 1 1049689.00 54017433.00
radix_sort 97.93 52966727 52900352 1 52900352.00 52966727.00
atoi 0.00 519 519 1 519.00 519.00
mta_clock_freq 0.00 24 24 4 6.00 6.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Memrefs Memrefs Calls
main 100.00 52966727 1
Callees
Function % Memrefs Memrefs Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

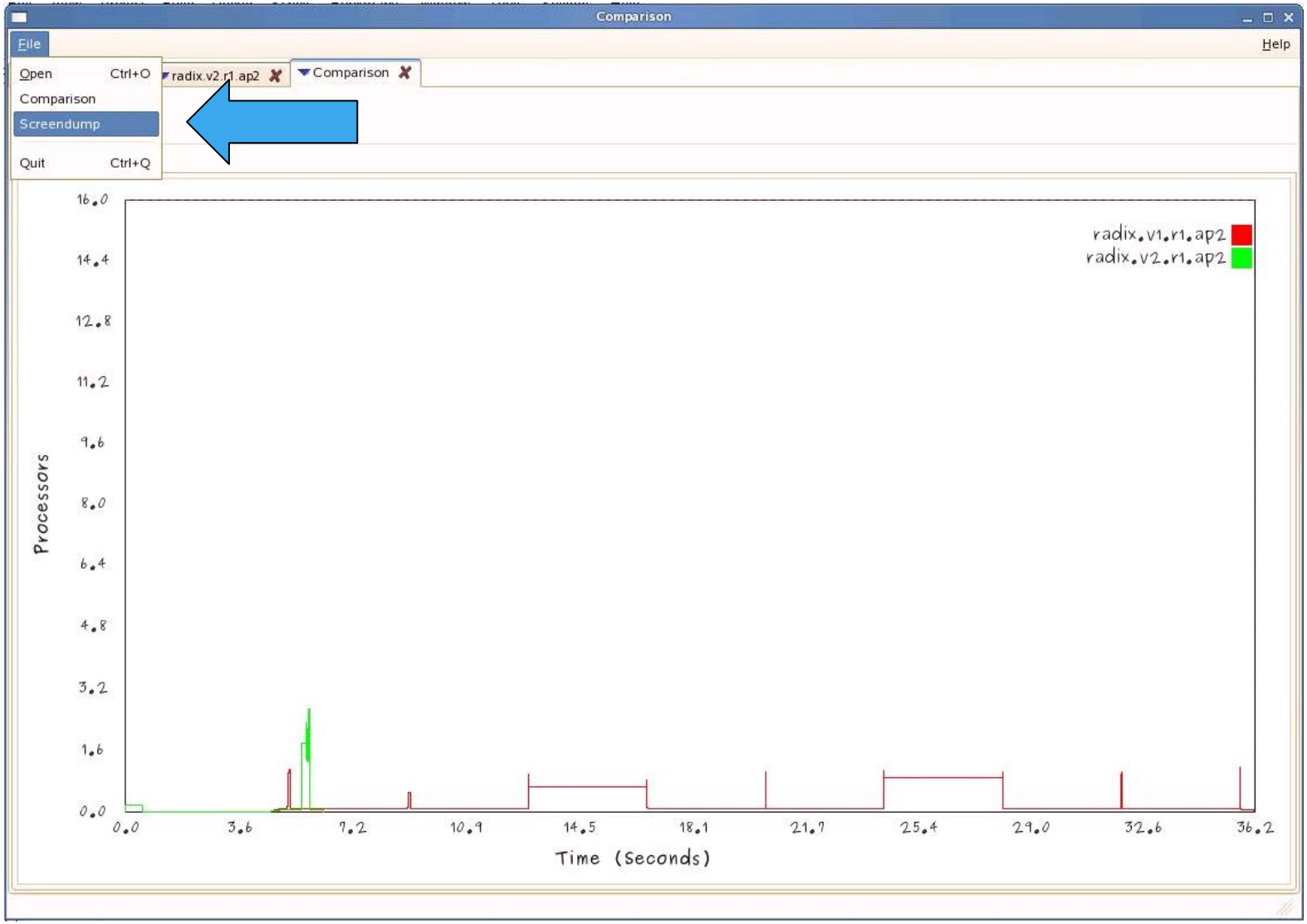

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Memrefs Total Memrefs Memrefs Calls Memrefs/Call Total Memrefs/Call
main 1.94 54017433 1049689 1 1049689.00 54017433.00
radix_sort 97.93 52966727 52900352 1 52900352.00 52966727.00
atoi 0.00 519 519 1 519.00 519.00
mta_clock_freq 0.00 24 24 4 6.00 6.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Memrefs Memrefs Calls
main 100.00 52966727 1
Callees
Function % Memrefs Memrefs Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

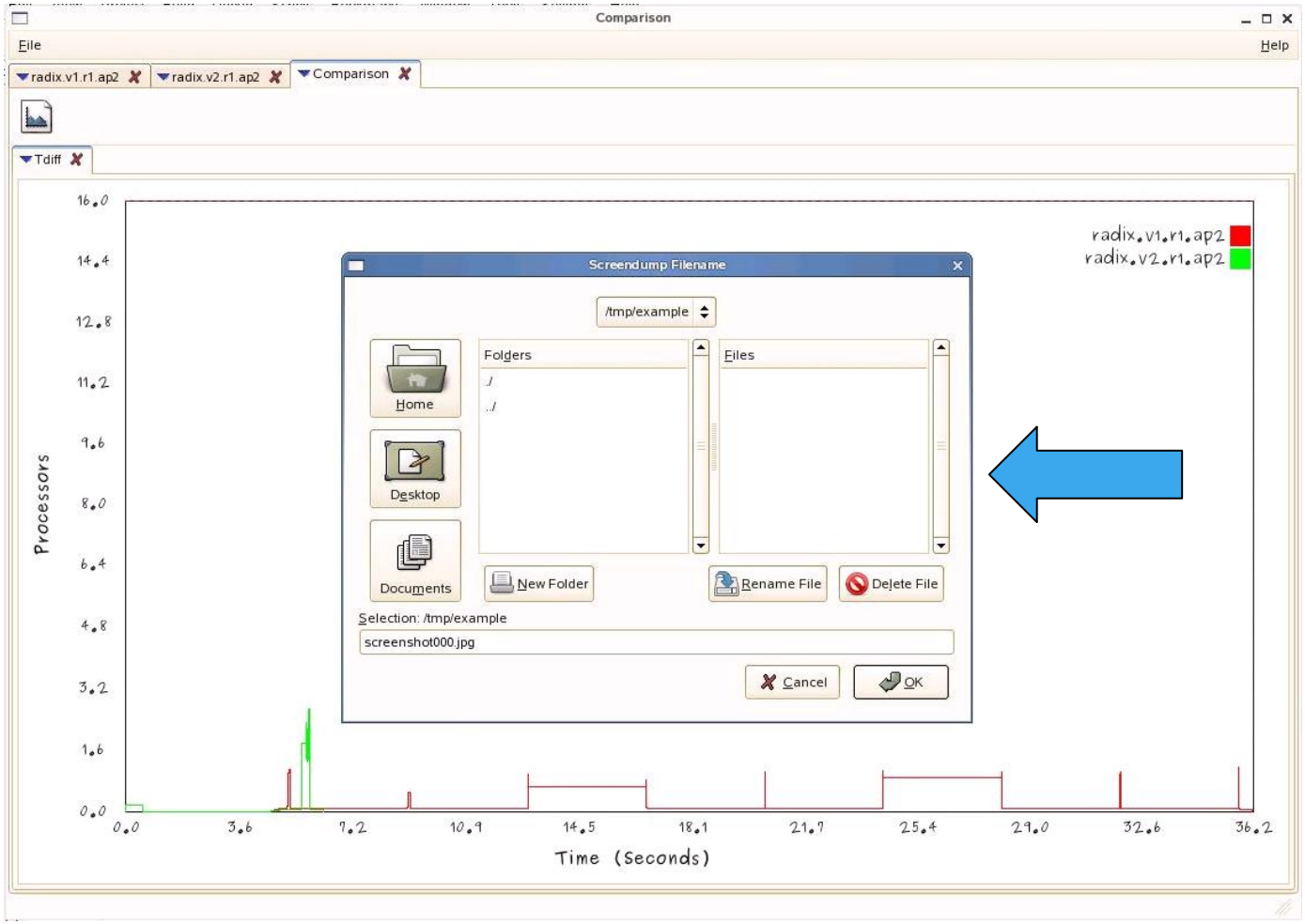

radix.v2.r1.ap2
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Tview
Canal
Bprof
Profiled 43.79% of execution (122.11M of approximately 278.84M issues), overheads: profiling 2.90%, parallel 7.36%, spill 0.00%, other 0.00%
Function % Memrefs Total Memrefs Memrefs Calls Memrefs/Call Total Memrefs/Call
main 1.94 54017433 1049689 1 1049689.00 54017433.00
radix_sort 97.93 52966727 52900352 1 52900352.00 52966727.00
atoi 0.00 519 519 1 519.00 519.00
mta_clock_freq 0.00 24 24 4 6.00 6.00
mta_get_num_teams 0.00 1 1 1 1.00 1.00
malloc 0.00 0 0 4 0.00 0.00
strtol 0.00 0 0 1 0.00 0.00
perror 0.00 0 0 0 0.00 0.00
Callers
Function % Memrefs Memrefs Calls
main 100.00 52966727 1
Callees
Function % Memrefs Memrefs Calls
__mta_trace 0.00 0 33
malloc 0.00 0 3
mta_get_num_teams 0.00 1 1
RT_Task::num_teams() 0.00 0 1
__crew_profile_barrier 0.00 0 1152
__free_vector 0.00 0 1
free 0.00 0 3
perror 0.00 0 0

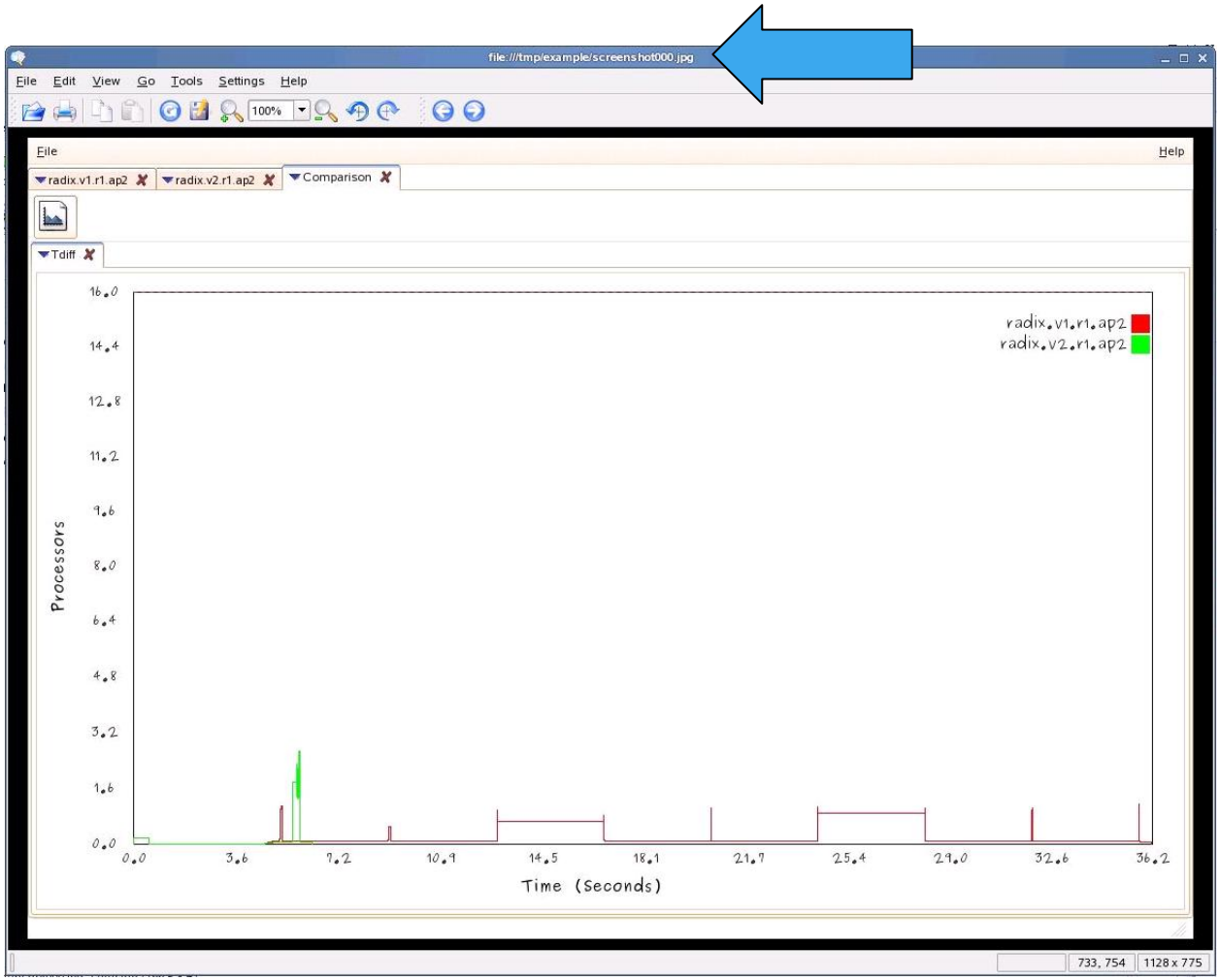

File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Comparison
Tdiff
radix.v1.r1.ap2
radix.v2.r1.ap2
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)

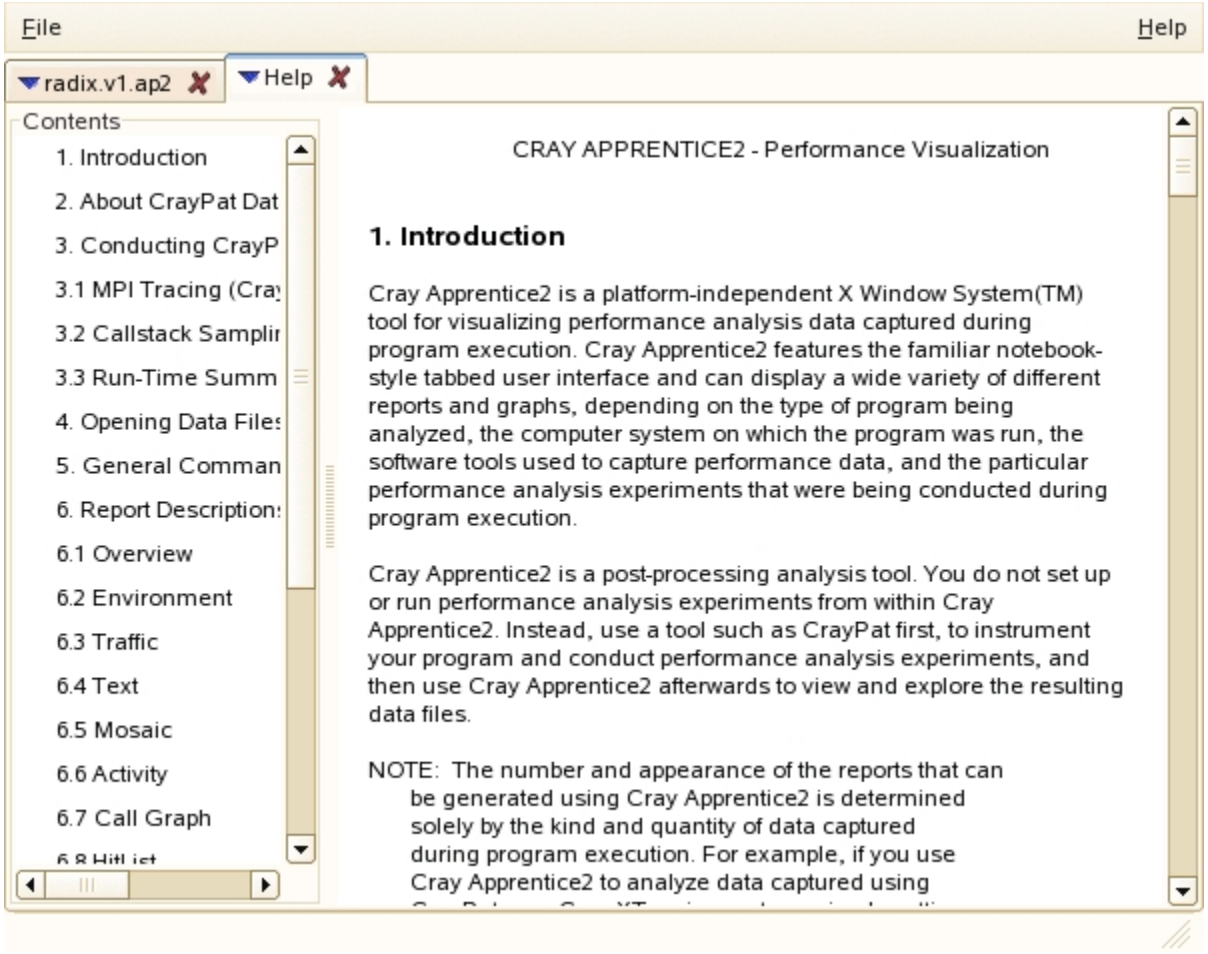

Comparison
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Comparison
Tdiff
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
radix.v1.r1.ap2
radix.v2.r1.ap2
Select Font
Family:
Purisa
Quadaptor
Quixotic
Radios in Motion
Radios in Motion Hard
Rafika
Style:
Medium
Italic
Bold
Bold Italic
Size:
12
10
11
12
13
14
Preview:
abcdefghijk ABCDEFGHIJK
Cancel
OK
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)


Comparison
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Comparison
Tdiff
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
radix.v1.r1.ap2
radix.v2.r1.ap2


Comparison
File
Help
Open Ctrl+O
Comparison
Screendump
Quit Ctrl+Q
radix.v2.r1.ap2
Comparison
radix.v1.r1.ap2
radix.v2.r1.ap2
Processors
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)


Comparison
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Comparison
Tdiff
Processors
Time (Seconds)
radix.v1.r1.ap2
radix.v2.r1.ap2
Screendump Filename
/tmp/example
Folders
Files
Home
Desktop
Documents
New Folder
Rename File
Delete File
Selection: /tmp/example
screenshot000.jpg
Cancel
OK


file:///tmp/example/screenshot000.jpg
File Edit View Go Tools Settings Help
100%
File
Help
radix.v1.r1.ap2
radix.v2.r1.ap2
Comparison
Tdiff
16.0
14.4
12.8
11.2
9.6
8.0
6.4
4.8
3.2
1.6
0.0
Processors
0.0
3.6
7.2
10.9
14.5
18.1
21.7
25.4
29.0
32.6
36.2
Time (Seconds)
radix.v1.r1.ap2
radix.v2.r1.ap2
733, 754
1128 x 775


File
Help
radix.v1.ap2
Help
Contents
1. Introduction
2. About CrayPat Dat
3. Conducting CrayP
3.1 MPI Tracing (Cra
3.2 Callstack Samplir
3.3 Run-Time Summ
4. Opening Data Files
5. General Comman
6. Report Description
6.1 Overview
6.2 Environment
6.3 Traffic
6.4 Text
6.5 Mosaic
6.6 Activity
6.7 Call Graph
CRAY APPRENTICE2 - Performance Visualization
1. Introduction
Cray Apprentice2 is a platform-independent X Window System(TM) tool for visualizing performance analysis data captured during program execution. Cray Apprentice2 features the familiar notebook-style tabbed user interface and can display a wide variety of different reports and graphs, depending on the type of program being analyzed, the computer system on which the program was run, the software tools used to capture performance data, and the particular performance analysis experiments that were being conducted during program execution.
Cray Apprentice2 is a post-processing analysis tool. You do not set up or run performance analysis experiments from within Cray Apprentice2. Instead, use a tool such as CrayPat first, to instrument your program and conduct performance analysis experiments, and then use Cray Apprentice2 afterwards to view and explore the resulting data files.
NOTE: The number and appearance of the reports that can be generated using Cray Apprentice2 is determined solely by the kind and quantity of data captured during program execution. For example, if you use Cray Apprentice2 to analyze data captured using

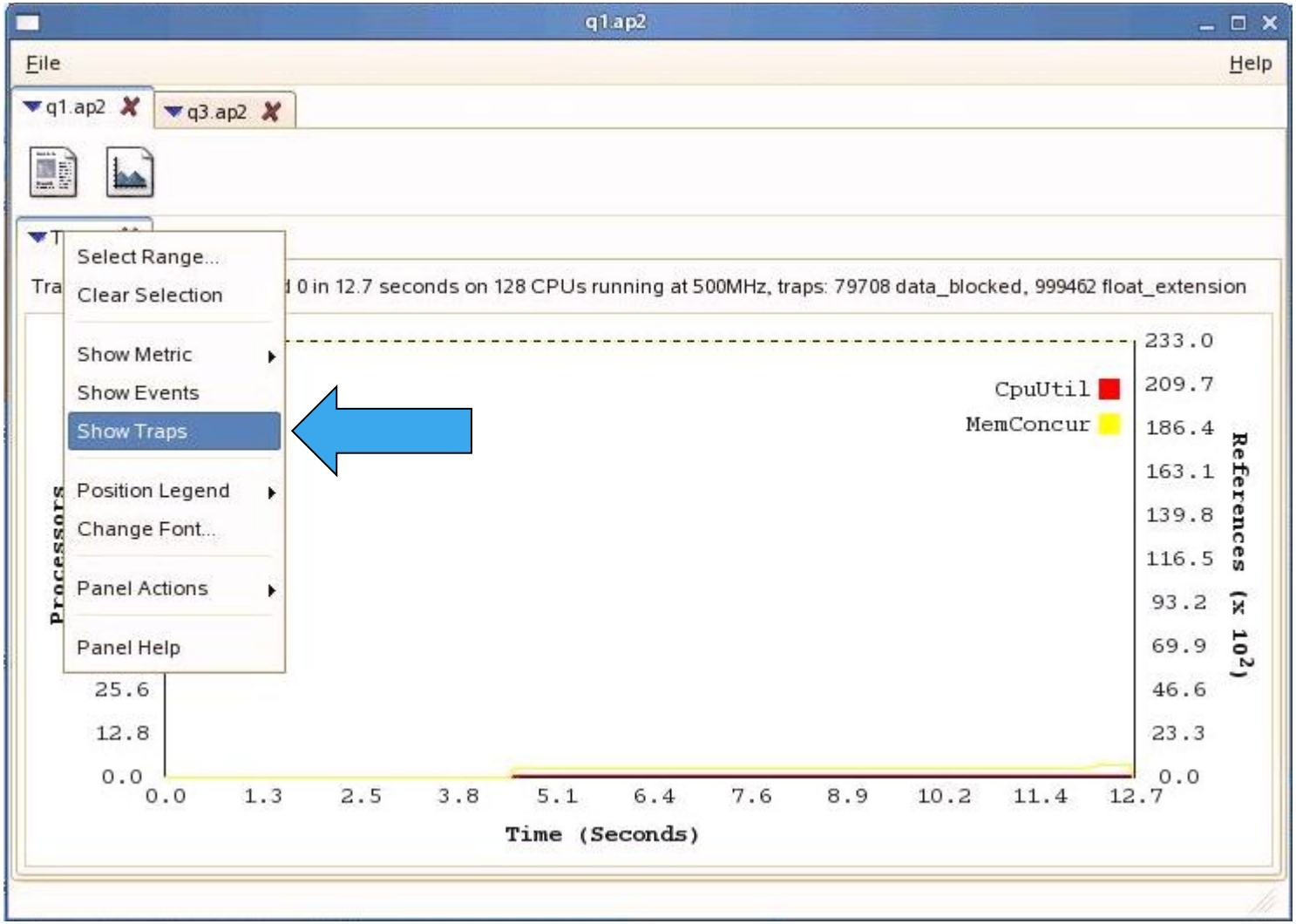
Workflow: Queue Append I

- **Write code**
  - **Append elements onto a queue like what might be done when looking for unvisited nodes in the adjacency lists for set of nodes during a parallelized BFS.**

```
unsigned sync k = 0;
#pragma mta assert parallel
#pragma mta use 100 streams
  for (unsigned i = 0; i < n; ++i) {
    for (unsigned j = 0; j < cnt[i]; ++j) {
      dst[int_fetch_add(&k, 1)] = src[idx[i]+j];
    }
  }
```

- **Compile, run, post-process, and run Apprentice2**


File
Help
q1.ap2
q3.ap2
Tview
Canal
Traced 286 events, dropped 0 in 12.7 seconds on 128 CPUs
, traps: 79708 data_blocked, 999462 float_extension
CpuUtil
MemConcur
Processors
References (x 10^2)
Time (Seconds)
128.0
115.2
102.4
89.6
76.8
64.0
51.2
38.4
25.6
12.8
0.0
233.0
209.7
186.4
163.1
139.8
116.5
93.2
69.9
46.6
23.3
0.0
0.0
1.3
2.5
3.8
5.1
6.4
7.6
8.9
10.2
11.4
12.7

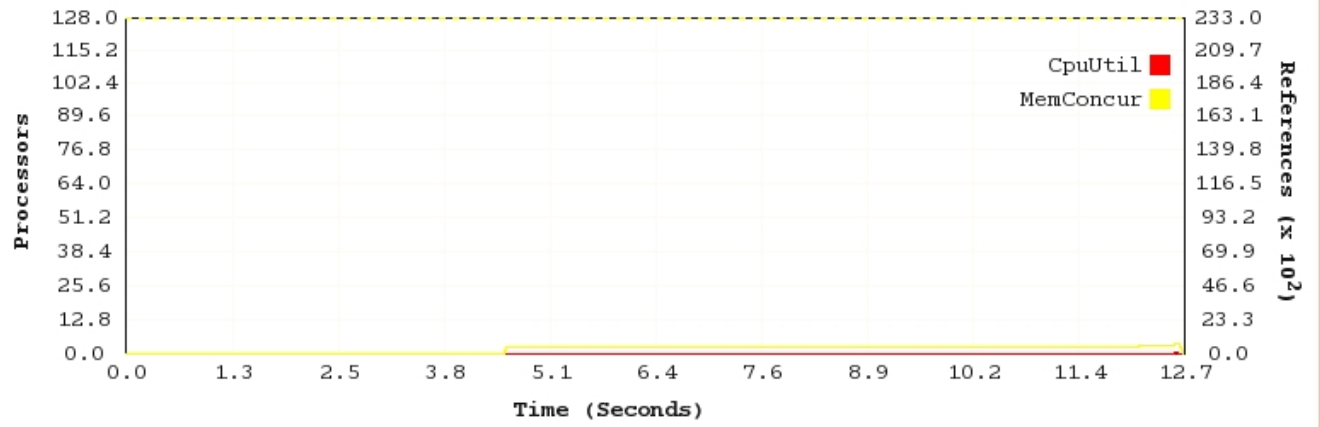

q1.ap2
File
Help
q1.ap2
q3.ap2
Select Range...
Clear Selection
Show Metric
Show Events
Show Traps
Position Legend
Change Font...
Panel Actions
Panel Help
0 in 12.7 seconds on 128 CPUs running at 500MHz, traps: 79708 data_blocked, 999462 float_extension
CpuUtil
MemConcur
233.0
209.7
186.4
163.1
139.8
116.5
93.2
69.9
46.6
23.3
0.0
References (x 10^2)
Processors
25.6
12.8
0.0
0.0
1.3
2.5
3.8
5.1
6.4
7.6
8.9
10.2
11.4
12.7
Time (Seconds)

File Help

q1.ap2 q3.ap2

Tview Canal

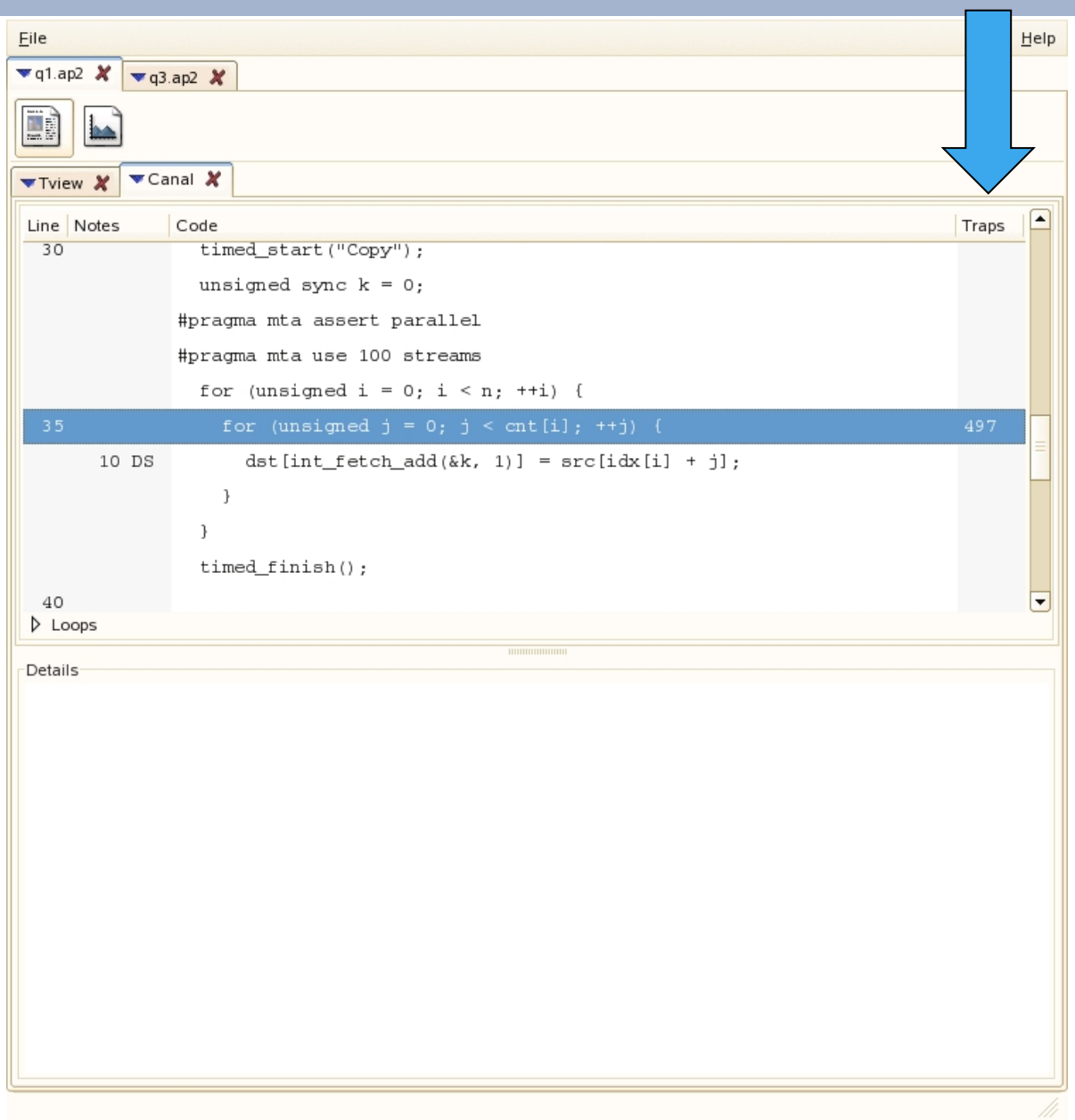
Traced 286 events, dropped 0 in 12.7 seconds on 128 CPUs running at 500MHz, traps: 79708 data_blocked, 999462 float_extension



Traps

Type	Data Result Code	Retry Op Code	Count	Rate	Time	Destination Register	Data Address	Program
FLOAT_EXTENSION	N/A	N/A	329	0.000000	12.475922	r0	0x0	0:
FLOAT_EXTENSION	N/A	N/A	186	0.000000	12.476206	r0	0x0	0:
	LATENCY_LIMIT	INT_FETCH_ADD	17	0.000000	12.584929	r15	0x400132bcf00	0:
DATA_BLOCKED	LATENCY_LIMIT	INT_FETCH_ADD	28	0.000000	12.587614	r15	0x400132bcf00	0:
DATA_BLOCKED	LATENCY_LIMIT	INT_FETCH_ADD	29	0.000000	12.600372	r15	0x400132bcf00	0:
DATA_BLOCKED	LATENCY_LIMIT	INT_FETCH_ADD	112	0.000000	12.643098	r15	0x400132bcf00	0:
DATA_BLOCKED	LATENCY_LIMIT	INT_FETCH_ADD	219	0.000000	12.644721	r15	0x400132bcf00	0:
DATA_BLOCKED	LATENCY_LIMIT	INT_FETCH_ADD	92	0.000000	12.645797	r15	0x400132bcf00	0:

Collate by program counter

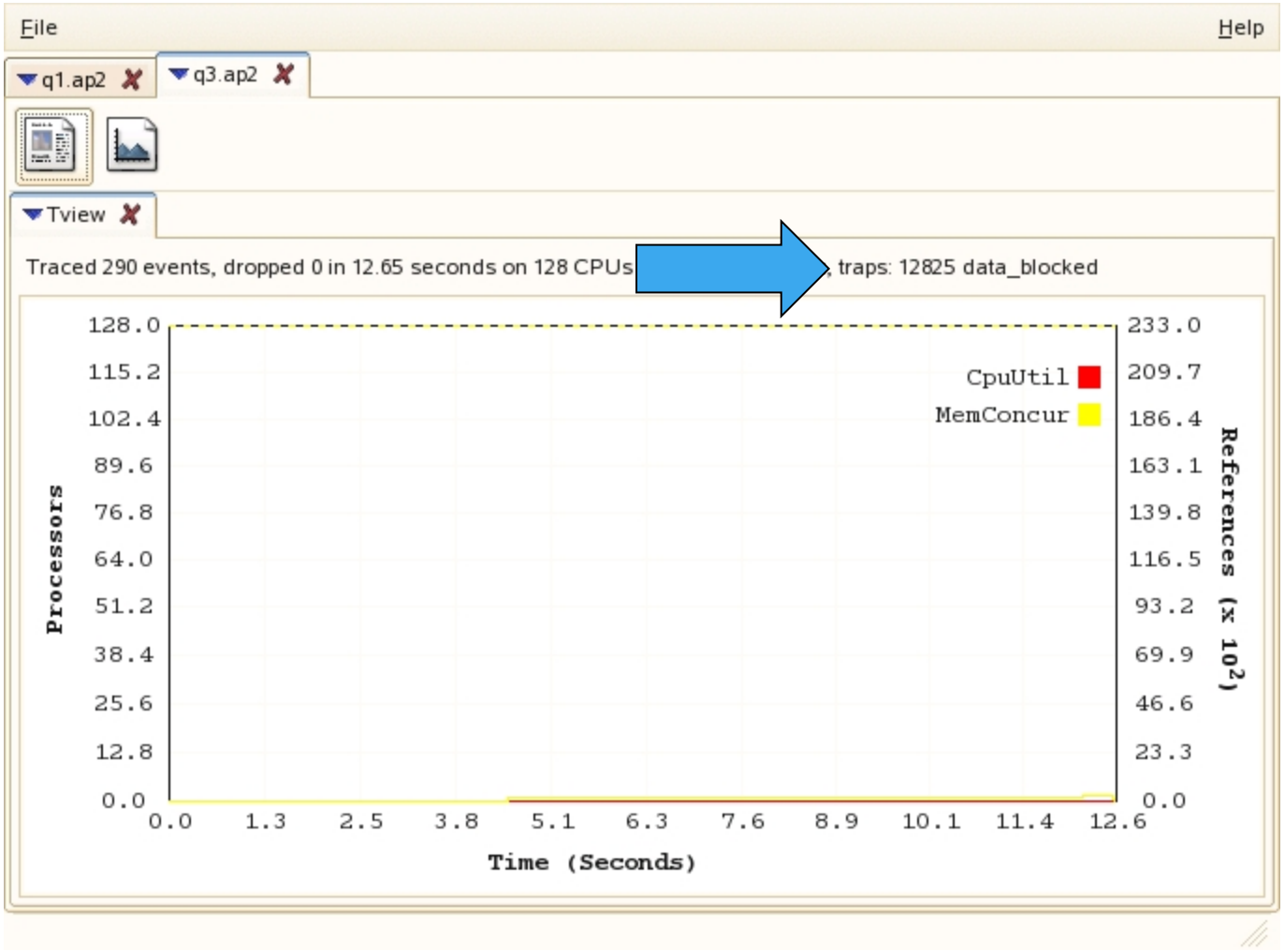

File
Help
q1.ap2
q3.ap2
Tview
Canal
Line
Notes
Code
Traps
30
timed_start("Copy");
unsigned sync k = 0;
#pragma mta assert parallel
#pragma mta use 100 streams
for (unsigned i = 0; i < n; ++i) {
35
for (unsigned j = 0; j < cnt[i]; ++j) {
497
10 DS
dst[int_fetch_add(&k, 1)] = src[idx[i] + j];
}
}
timed_finish();
40
Loops
Details

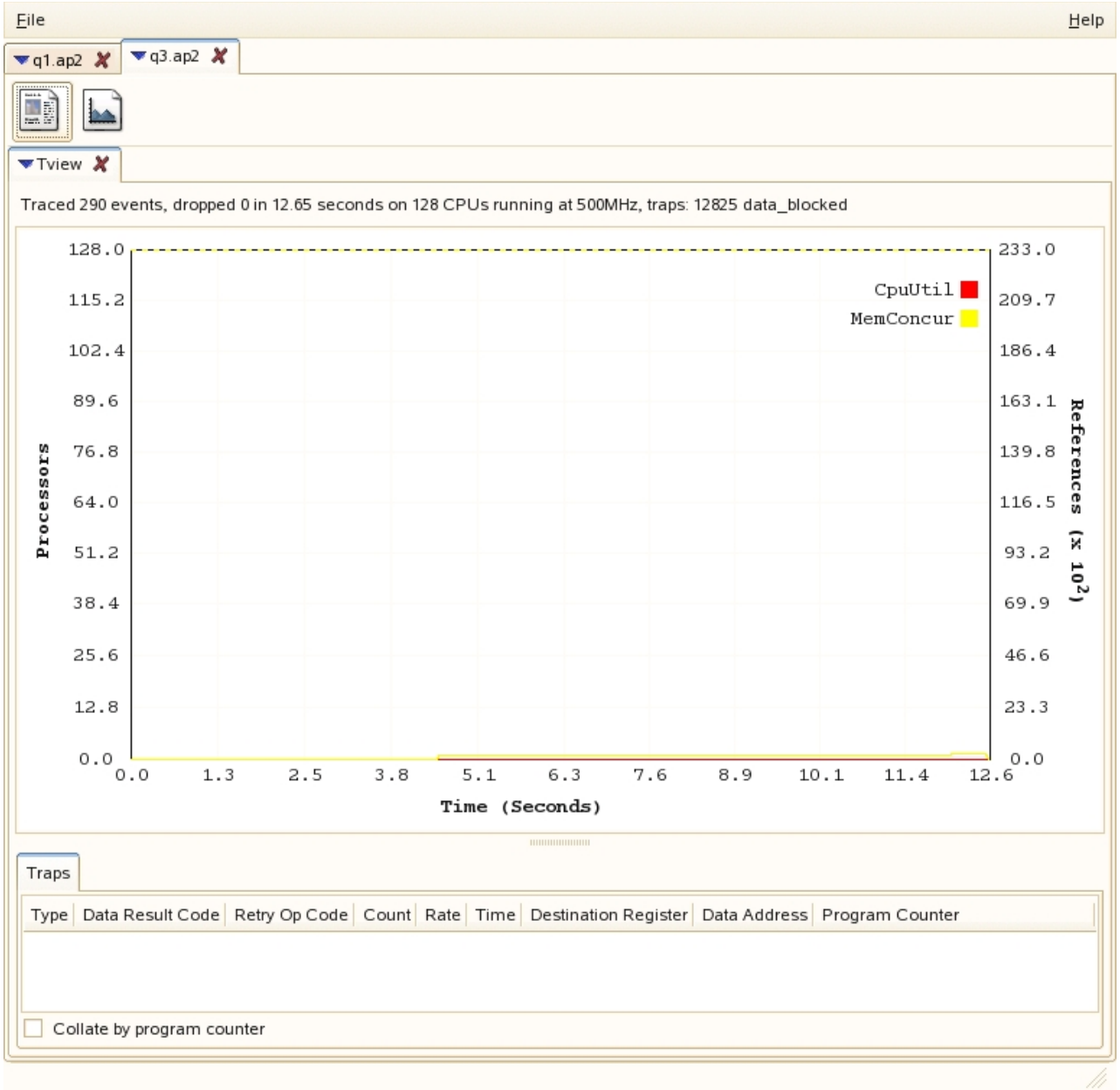
Workflow: Hotspot Queue II

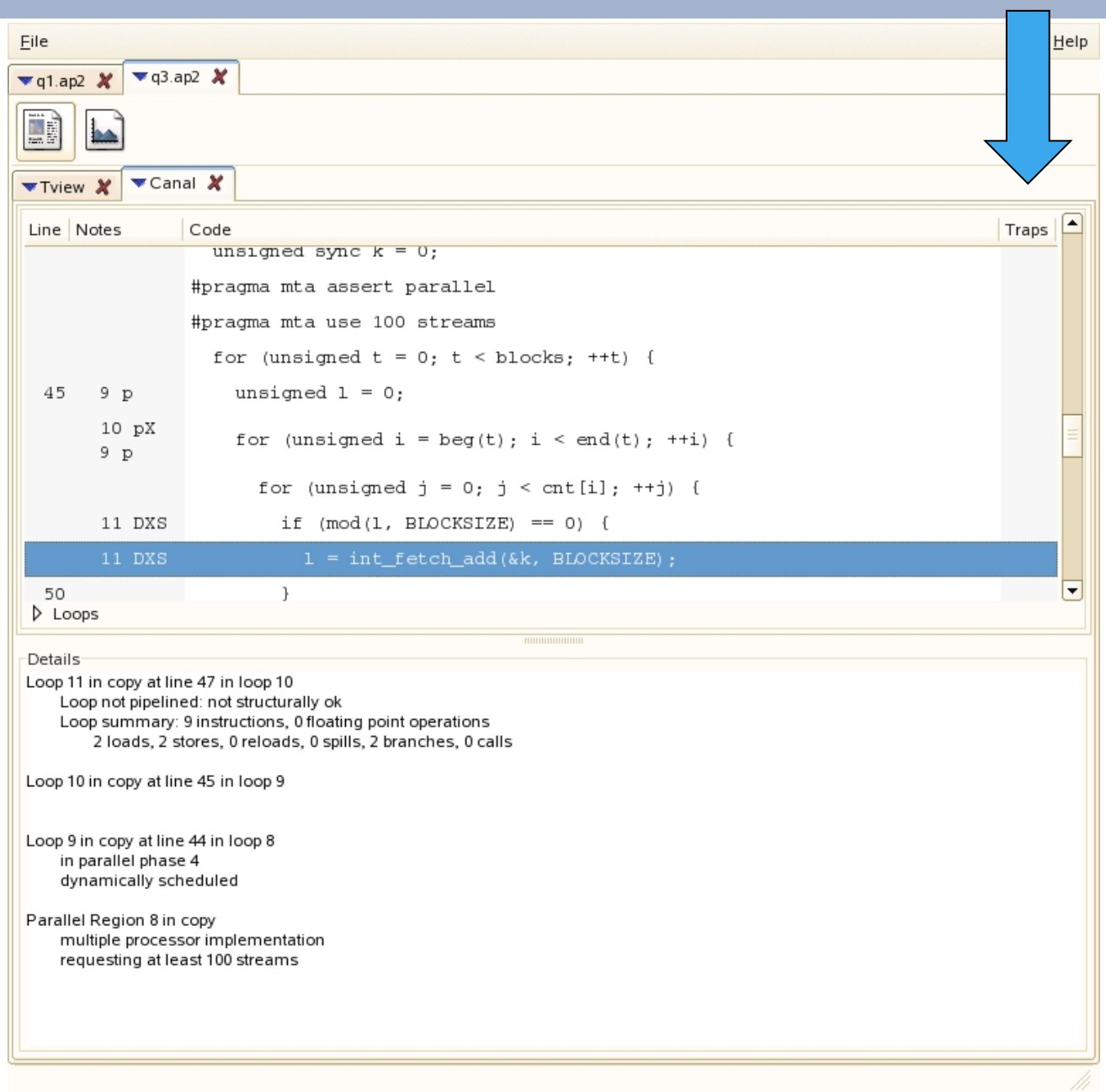
- Modify code
  - Grab multiple input and output elements at a time, allowing blanks

```
for (unsigned t = 0; t < blocks; ++t) {
  unsigned l = 0;
  for (unsigned i = beg(t); i < end(t); ++i) {
    for (unsigned j = 0; j < cnt[i]; ++j) {
      if (mod(l, BLOCKSIZE) == 0) {
        l = int_fetch_add(&k, BLOCKSIZE);
      }
      dst[l++] = src[idx[i] + j];
    }
  }
}
```

- Compile, run, post-process, and run Apprentice2


File
Help
q1.ap2
q3.ap2
Tview
Traced 290 events, dropped 0 in 12.65 seconds on 128 CPUs
traps: 12825 data_blocked
128.0
115.2
102.4
89.6
76.8
64.0
51.2
38.4
25.6
12.8
0.0
Processors
233.0
209.7
186.4
163.1
139.8
116.5
93.2
69.9
46.6
23.3
0.0
References (x 10^2)
CpuUtil
MemConcur
0.0
1.3
2.5
3.8
5.1
6.3
7.6
8.9
10.1
11.4
12.6
Time (Seconds)


File
Help
q1.ap2
q3.ap2
Tview
Traced 290 events, dropped 0 in 12.65 seconds on 128 CPUs running at 500MHz, traps: 12825 data_blocked
128.0
115.2
102.4
89.6
76.8
64.0
51.2
38.4
25.6
12.8
0.0
Processors
233.0
209.7
186.4
163.1
139.8
116.5
93.2
69.9
46.6
23.3
0.0
References (x 10²)
CpuUtil
MemConcur
0.0
1.3
2.5
3.8
5.1
6.3
7.6
8.9
10.1
11.4
12.6
Time (Seconds)
Traps
Type
Data Result Code
Retry Op Code
Count
Rate
Time
Destination Register
Data Address
Program Counter
Collate by program counter


File
Help
q1.ap2
q3.ap2
Tview
Canal
Line
Notes
Code
Traps
unsigned sync k = 0;
#pragma mta assert parallel
#pragma mta use 100 streams
for (unsigned t = 0; t < blocks; ++t) {
45 9 p
unsigned l = 0;
10 pX
9 p
for (unsigned i = beg(t); i < end(t); ++i) {
for (unsigned j = 0; j < cnt[i]; ++j) {
11 DXS
if (mod(l, BLOCKSIZE) == 0) {
11 DXS
l = int_fetch_add(&k, BLOCKSIZE);
50
}
Loops
Details
Loop 11 in copy at line 47 in loop 10
Loop not pipelined: not structurally ok
Loop summary: 9 instructions, 0 floating point operations
2 loads, 2 stores, 0 reloads, 0 spills, 2 branches, 0 calls
Loop 10 in copy at line 45 in loop 9
Loop 9 in copy at line 44 in loop 8
in parallel phase 4
dynamically scheduled
Parallel Region 8 in copy
multiple processor implementation
requesting at least 100 streams

Source Code Trace Controls

`#pragma mta trace [on | off | default]`

- Doesn't work unless compiled with the –trace flag

`#pragma mta trace level` *`n`*

- Trace only functions that contain at least *n* lines
- Affects the rest of the source code in this file

`#pragma mta trace "`*`string that identifies this point in the source code`*`"`

- Resource usage data is recorded at this point and associated with this character string in the Traceview textual output

Managing very large trace/ap2 files

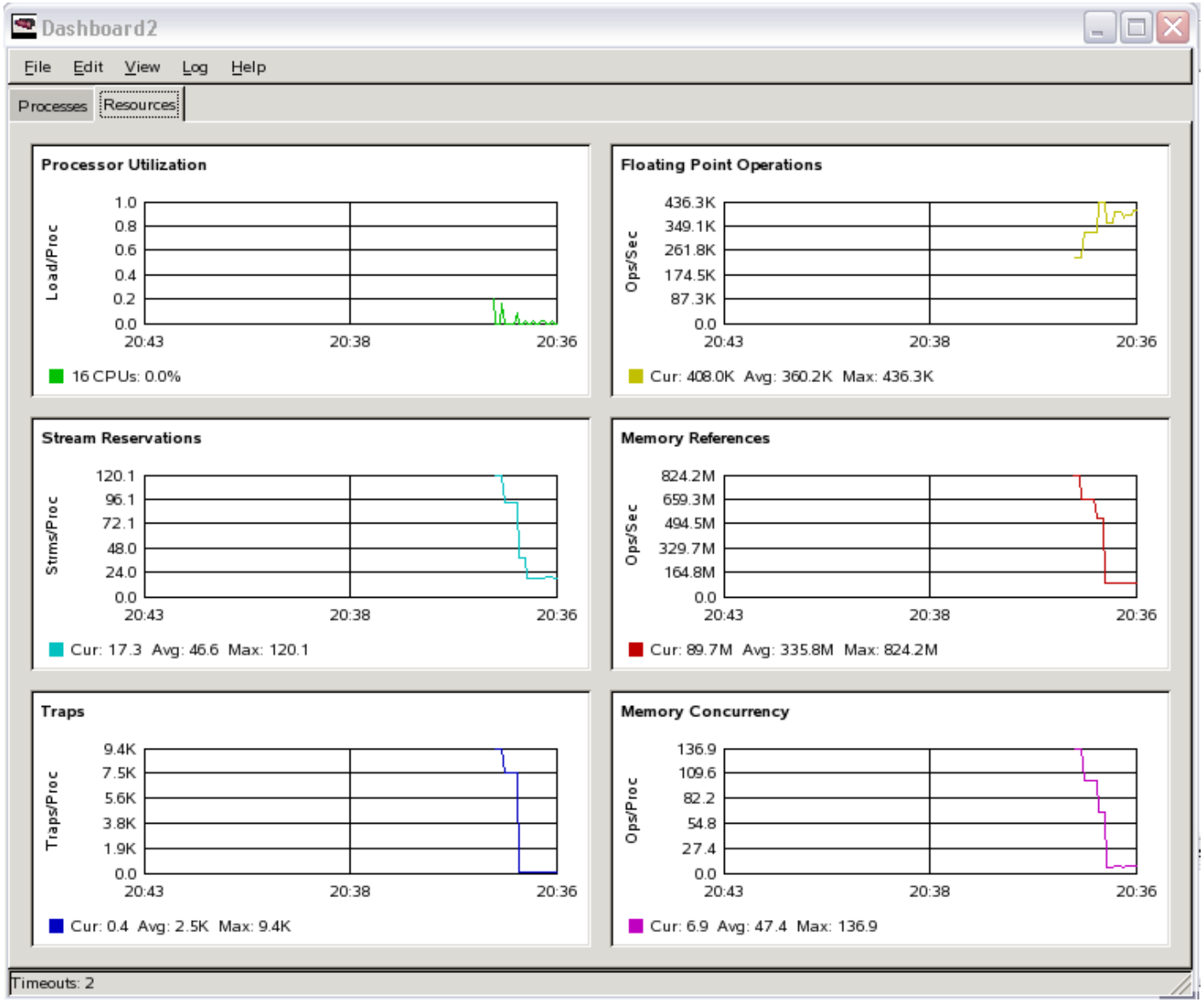
- Created mechanism using temporary to help offset memory usage on the login nodes
  - XMT login nodes lack the ability to swap
- Root name for temp files set by environment variable
  - export APP2_SWAPFILE = /mnt/lustre/users/app2
  - Will generate files with names like
    - /mnt/lustre/users/app2.xxxxxx
- Temp files cleaned when exiting Apprentice2

Summary

- Apprentice2
  - GUI based system allows rapid assimilation of information
  - Interaction between reports assists problem detection
- Canal Report
  - Establishes a dialogue with the compiler that allows the creation of highly parallelized and optimized code
- Tview Report
  - Quickly identify underperforming code sections
  - Visualize resource utilization and concurrency
  - Pinpoint memory hotspots
- Bprof Report
  - Target specific functions running the least efficiently

Additional Tools and Approaches

Dash: a Real-Time Resource Monitor



Accessing the Hardware Counters

```
unsigned issues, memrefs, concur, streams, traps, retries;
mta_reserve_task_event_counter( RT_ANY_COUNTER, RT_TRAP );
 /* other counters are available without reserving them */
  issues = mta_get_task_counter(RT_ISSUES);
  memrefs= mta_get_task_counter(RT_MEMREFS);
  concur= mta_get_task_counter(RT_CONCURRENCY);
  streams= mta_get_task_counter(RT_STREAMS);
  traps = mta_get_task_counter(RT_TRAP);
  retries = mta_get_task_counter(RT_MEM_RETRY);
  /* (continued on next slide) */
```

Accessing the Hardware Counters

```
double start_time = timer();

int result = bigParallelFunction();

double elapsed_time = start_time - timer();


issues = mta_get_task_counter(RT_ISSUES) - issues;

memrefs= mta_get_task_counter(RT_MEMREFS) - memrefs;

concur= mta_get_task_counter(RT_CONCURRENCY) - concur;

streams= mta_get_task_counter(RT_STREAMS) - streams;

traps = mta_get_task_counter(RT_TRAP) - traps;

retries = mta_get_task_counter(RT_MEM_RETRY) - retries;

/* printf() probably goes here */
```

Example: Using Hardware Counters

- **Customer was observing significant performance difference between the two loops.**
  - **Loop 1:**

```
for (i = 0; i < no_of_edges; i++) {
    ends[i] = endpoints[i] + 1;
}
```

  - **Loop 2:**

```
for(i = 0; i <  no_of_edges; i++) {
    endpoints[i] = ends[i] % BILLION;
}
```

- **Significant point:**
  - **Endpoints is an array of unsigned integers, initialized using prand_int.**
  - **Elements of endpoints are then randomly distributed 64-bit unsigned integers over the range of all possible 64-bit unsigned integers.**

Modulo Instrumentation Example

Loop1: time copying to ends 6.654852e-02

```
for (i = 0; i < no_of_edges; i++) {
     ends[i] = endpoints[i] + 1;
}
```

ticks: 33257303, secs: 0.066515, issues: 104509017, memrefs: 101770916, traps: 0, retries: 717757, concurrency: 2542, streams: 755

Loop2: time modulo 4.534740e+01

```
for(i = 0; i <  no_of_edges; i++) {
    endpoints[i] = ends[i] % BILLION;
}
```

ticks: 22673682155, secs: 45.347364, issues: 32993662421, memrefs: 15585715642, traps: 49976693, retries: 18913196115, concurrency: 811, streams: 610

Modulo Instrumentation Example

XMT only provides a 53-bit integer divide

Details:

- If either `x` or `y` is more than 53 bits, the float extension will be raised and the trap handler will complete the operation successfully.
- Sign extension is handled correctly, so for example, `INV_DIV_CHOP/INV_DIV_FLOOR` support values in the range $[-2^{53}, 2^{53}-1]$, whereas `UNS_DIV` supports values in the range $[0, 2^{53}-1]$.

Modulo Instrumentation Example

The performance issue was that the modulo operator was being applied to integers generated to be random across all 64-bit unsigned integers.

Modified code snippet which avoids 64-bit divide

```
unsigned int mask = (1 << 53) - 1;
#pragma mta assert nodep *endpoints
#pragma mta assert nodep *ends
for (i = 0; i < no_of_edges; i++) {
     endpoints[i] = (ends[i] & mask) % N2;
}
```

Modulo Instrumentation Example

Loop1: time copying to ends 6.654919e-02

```
for (i = 0; i < no_of_edges; i++) {
    ends[i] = endpoints[i] + 1;
}
```

ticks: 33258200, secs: 0.066516, issues: 104491617, memrefs: 101756261,
traps: 0, retries: 705688, concurrency: 2541, streams: 754

Loop2: time modulo 8.084661e-02

```
unsigned int mask = (1 << 53) - 1;
for (i = 0; i < no_of_edges; i++) {
     endpoints[i] = (ends[i] & mask) % N2;
}
```

ticks: 40407611, secs: 0.080815, issues: 155276319, memrefs: 102072724,
traps: 0, retries: 673585, concurrency: 1939, streams: 720

Looking at Assembly Code

- From the command line

  `dis executableName`

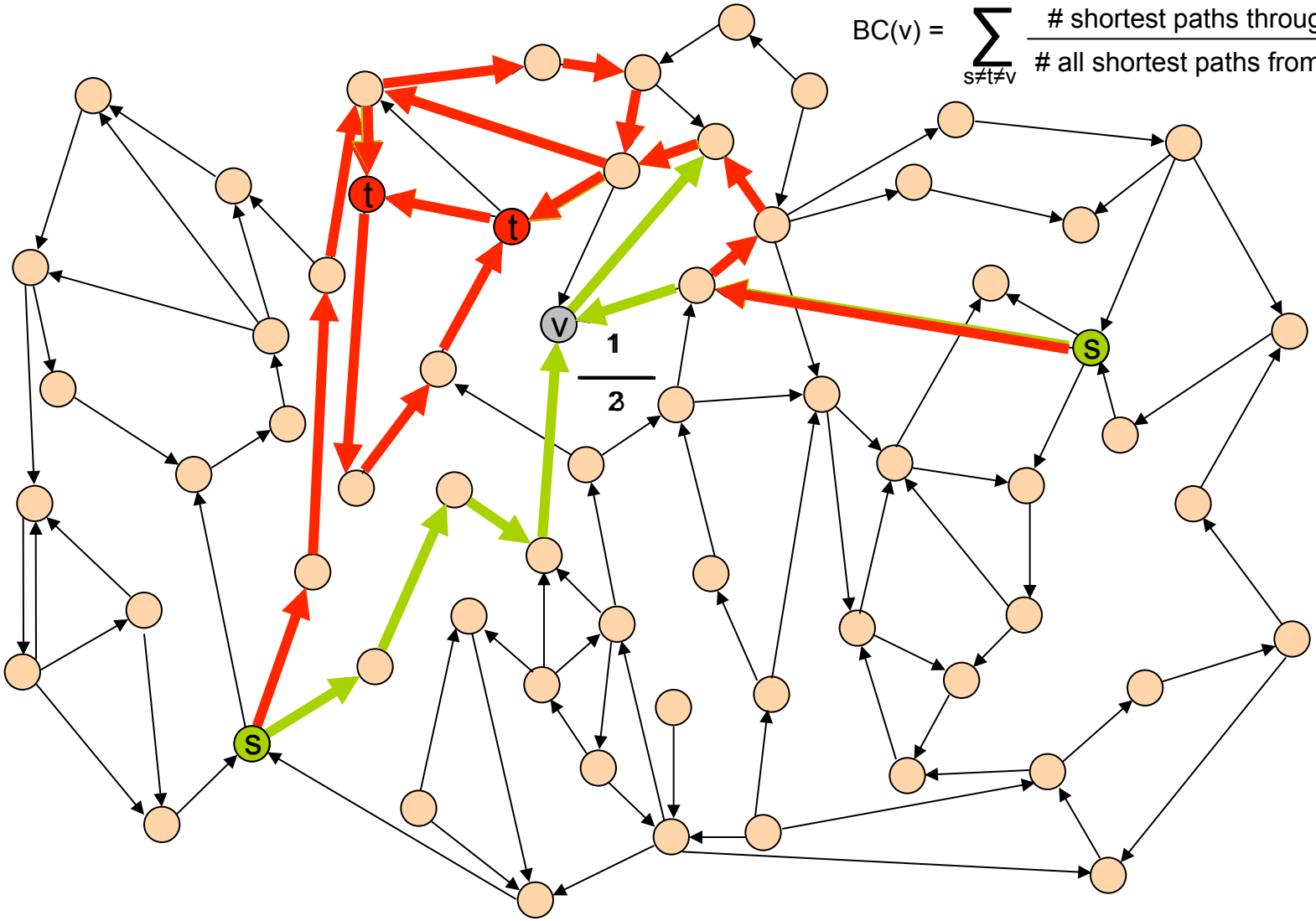
- From within mdb

  `mdb> disass`

```
0x004ac9ec      (inst 2 (STORE_DISP r25 r1 (* 8 2)) (NOP))
0x004ac9ed      (inst 1 (STORE_DISP r19 r1 (* 8 9)) (NOP))
0x004ac9ee      (inst 0 (STORE_DISP r21 r1 (* 8 12)) (NOP))
0x004ac9ef      (inst 0 (NOP) (INT_ADD_IMM r30 r0 15) (CLOCK r26 r0))
0x004ac9f0      (inst 0 (REG_LOAD_DISP r25 r31 (* 8 10)) (BIT_AND r30 r26 r30))
0x004ac9f1      (inst 0 (REG_LOAD_DISP r3 r31 (* 8 10)) (INT_ADD_IMM r25 r25 104))
0x004ac9f2      (inst 2 (LOAD_DISP r28 r31 (* 8 19)) (INT_ADD_IMM r3 r3 96))
0x004ac9f3      (inst 0 (LOAD r25 r25) (INT_ADD_IMM r29 r0 216) (REG_MOVE r10 r9))
0x004ac9f4      (inst 3 (LOAD r3 r3) (TARGET_RESTORE t5 r25) (REG_MOVE r5 r27))
0x004ac9f5      (inst 0 (LOAD_AC_INDEX r28 r28 ( (fe_control FE_FUTURE)) r30) (TARGET_DISP t1…
```

SSCA #2, Kernel 4,
Betweenness Centrality

Betweenness Centrality of a Vertex

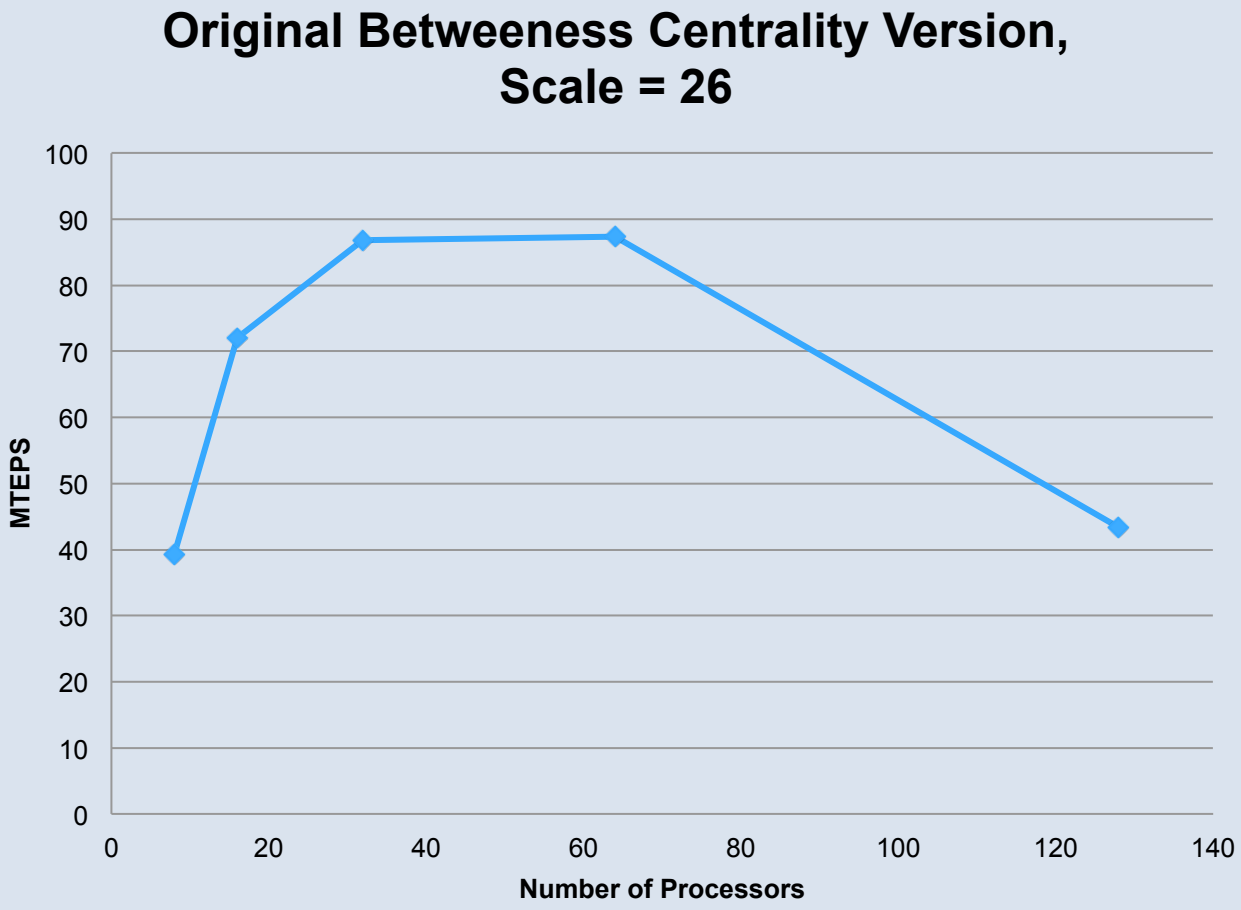


Our Original Implementation

- **Large data structures => sequential outer loop**

```
/* Use |Vs| nodes to compute centrality values */
  for (s = 0; (s < NV) && (Vs > 0); s ++) {
  …
#pragma mta assert no dependence
      for (j = QHead[nQ - 1]; j < QHead[nQ]; j++) {
          …
          int myStart = start[v];
          int myEnd   = start[v + 1];
#pragma mta assert no dependence
          for (k = myStart; k < myEnd; k++) {
              …
```

Original Version Didn't Scale Well



Note: MTEPS=millions of traversed edges per second, estimate defined as #vertices * 7 / execution time.

Trying to Scale Better

- With larger systems, we thought we could run a few iterations of the outermost loop in parallel
- First attempt:

```
#define BFS_THREADS  16
…
#pragma mta assert parallel
  for(num_threads=0; num_threads < BFS_THREADS; num_threads ++){
for(;;) {
          start_vertex = int_fetch_add(&Vs_ptr,1);
          if (start_vertex > Vs -1) break;
          …

#pragma mta assert no dependence
          for (j = QHead[nQ - 1]; j < QHead[nQ]; j++) {
```

Resulting Canal Listing

- **Performance was horrible.**
- **Canal:**

```
...
7 pXX       |   for (j = QHead[nQ - 1]; j < QHead[nQ]; j++) {
...
```

- **What happened:**
  - **Compiler assumed there was sufficient parallelism in the outer loop.**

Second Try

- Using “loop future” parallelism

```
#pragma mta assert parallel
#pragma mta loop future
for(num_threads=0; num_threads < BFS_THREADS; num_threads ++){
…
```

- Performance was bad.
- Canal output:

```
Parallel region   7 in cenTrality in loop 6
          Single processor implementation
```

- Parallelism was confined within BFS_THREADS processors.

Third Approach

- Use “future” variables

```
#define BFS_THREADS 16
  future int thread_id[BFS_THREADS];
  …
// Spawn off futures to run independent BFS processes
  for (num_threads=0; num_threads < BFS_THREADS; num_threads++) {
     future thread_id[num_threads](num_threads, G, BC,
                                   Vs, &Vs_ptr, permV, bfs_counter) {
        Process_cenTrality(G, BC, Vs, &Vs_ptr, …
        …
  for (num_threads=0; num_threads < BFS_THREADS; num_threads++) {
     touch (&thread_id[num_threads]);
  }
 …
```

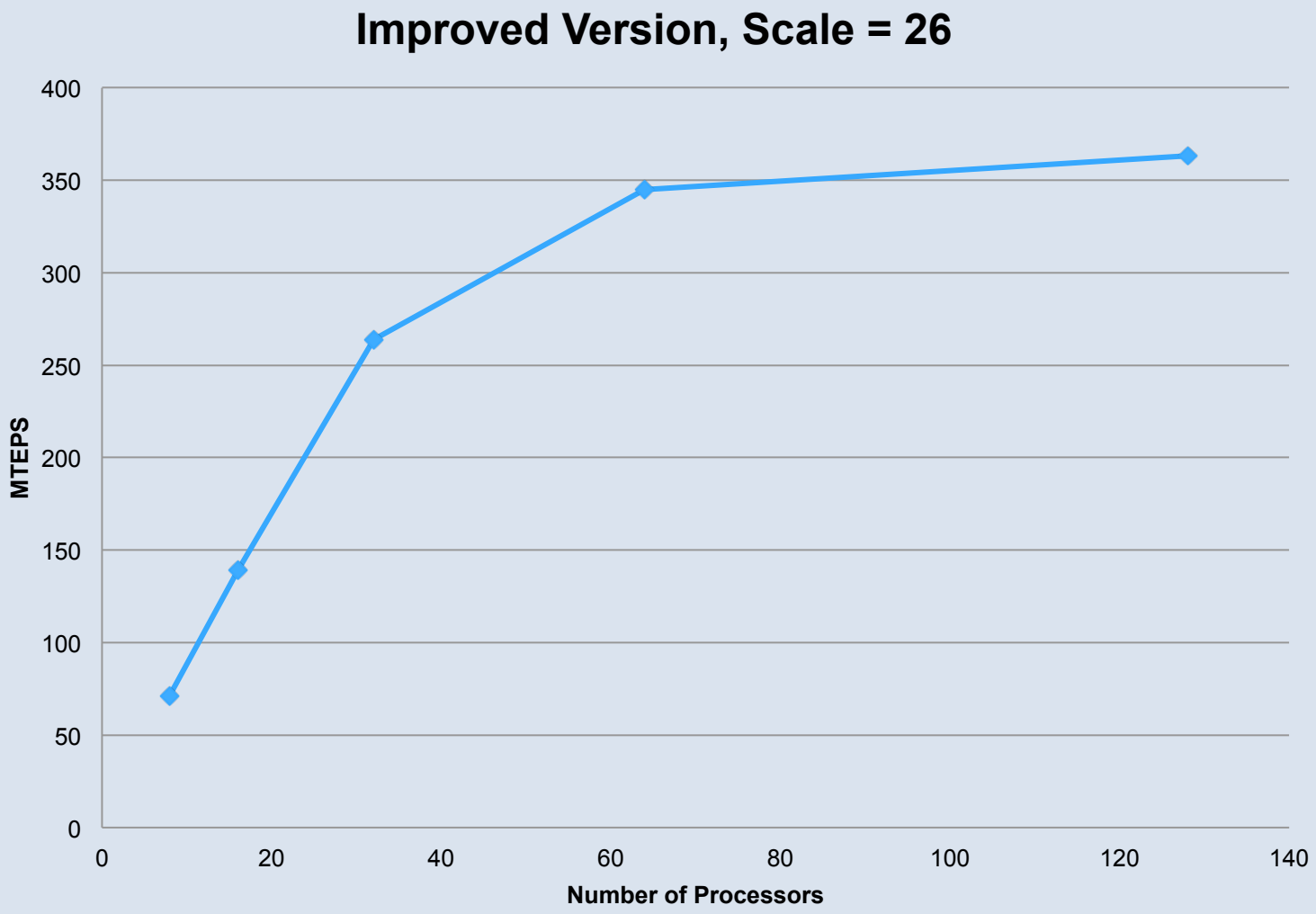
Additional tuning

#pragma mta max *n* processors

- Limits the number of processors used on a multi-processor loop to *n*
- *n* must be a compile-time integer constant > 0
- for collapsible loop nests, the max processors value for the collapsed loop is the same as that specified for the outer loop

```
 /* Use at most 4 processors.  */
#pragma mta max 4 processors
 for(i = 0; i < size; i++) {
    array[i] += array[i] + (size + i);
 }
```

Much Improved Performance and Scaling



Lessons Learned

- We used future variable parallelism because
  - The amount of parallelism in the outermost loop was small
  - The amount of parallelism in the inner loops was large and dynamic
- We used #pragma mta max n processor to control how wide each of the inner parallel loops could grow
  - Better load balancing across outer loops
  - Avoid hotspotting in the Queues

This also works

- Use “mta loop future” pragma on outer loop, but have inner loop inside subroutine call
  - Loop parallelism inside Process_cenTrality subroutine call is now multi-processor
  - OK to do it this way, but not the intended use of the loop future pragma. More targeted for large loop bounds.

```
#define BFS_THREADS 16
  …
// Outer loop to run independent BFS processes
#pragma mta loop future
  for (num_threads=0; num_threads < BFS_THREADS; num_threads++) {
      Process_cenTrality(G, BC, Vs, &Vs_ptr, …
          …
   }
```

Parallelism-Limiting Pragmas

#pragma mta max *n* processors

- Limits the number of processors used on a multiprocessor loop to *n*
- *n* must be a compile-time integer constant > 0
- for collapsible loop nests, the max processors value for the collapsed loop is the same as that specified for the outer loop

```
 /* Use at most 4 processors.  */
#pragma mta max 4 processors
 for(i = 0; i < size; i++) {
    array[i] += array[i] + (size + i);
 }
```

Parallelism-Limiting Pragmas

#pragma mta max concurrency *c*

- **Limits the number of processors used by a multiprocessor loop to *max(1, c/<num_streams_per_processor>)*, where *<num_streams_per_processor>* is the number of streams the compiler requests for each processor used by the parallel loop.**
- **Limits the number of streams used by a single processor parallel loop to *min(c, <max_streams_per_processor>)***
- ***c* is a compile-time integer constant > 0**

```
/* Use at most 512 streams across all processors.  */
#pragma mta max concurrency 512
  for(i = 0; i < size; i++) {
     array[i] += array[i] + (size + i);
  }
```

Parallelism-Limiting Pragmas

- **Using them together…**

```
/* Use at most 512 streams across all processors or
   at most 8 processors, whichever is smaller      */
#pragma mta max concurrency 512
#pragma mta max 8 processors
 for(i = 0; i < size; i++) {
     array[i] += array[i] + (size + i);
 }
```

- **Loop future loops can only use the max concurrency pragma:**

```
/* Create at most 512 futures.  */
#pragma mta loop future
#pragma mta max concurrency 512
 for(i = 0; i < size; i++) {
     array[i] += array[i] + (size + i);
 }
```

New Pragmas in XMT PE Release 1.4 (1)

```
#pragma mta for all streams
{
   …
}
```

- starts a parallel region
- executes the statement or block of statements exactly once for each stream allocated to the region
- acts like an “assert parallel” pragma
- can be used in conjunction with “use *n* streams” (but no guarantee that many will be allocated)

```
#pragma mta use 100 streams
#pragma mta for all streams
{  //do parallel stuff
}
```

New Pragmas in XMT PE Release 1.4 (2)

#pragma mta for all streams *i* of *n*

- Sets *n* to the total number of streams executing the region
- Variable *i* is a unique per-stream identifier; 0 ≤ *i* ≤ *n*-1

```
int istr, ntotal;
int check_in_array[SOME_BIG_NUMBER];

for( istr=0; istr<SOME_BIG_NUMBER; istr++ )
   check_in_array[istr] = 0;

#pragma mta for all streams istr of ntotal
{  check_in_array[istr] = 1;
   printf("Stream %d of %d checked in.\n", istr, ntotal);
}
```

Generating Graphs on the XMT

Generating synthetic graphs

- Random graphs
- R-MAT graphs

Random Graph

- Or Erdös-Renyi graph
- Given the number of vertices *n*, average number of out-edges per vertex (out-degree) *x*,
- Return a randomly-generated graph with *n* vertices with a uniformly-distributed out-degree with average *x*
- Random graphs are pretty "well-behaved"
  - Partition reasonably well

What Our Random Graph Generator Does

- SSCA #2 Handout 1
- Given desired # vertices (int), and desired average out-degree (double), generates random graph
- How it works:
  - Generate array, length `numEdges`, with random vertex IDs in it

```
randNeighbors(Neighbors, nN, numEdges,(double*)Marked, nN);
```

  - Histogram the IDs

```
numNeighbors[Neighbors[i]]++;
```

  - This decides how many out-edges each vertex has.
  - Now generate another array of `numEdges` random vertex IDs

```
randNeighbors(Neighbors, nN, numEdges,(double*)Marked, nN);
```

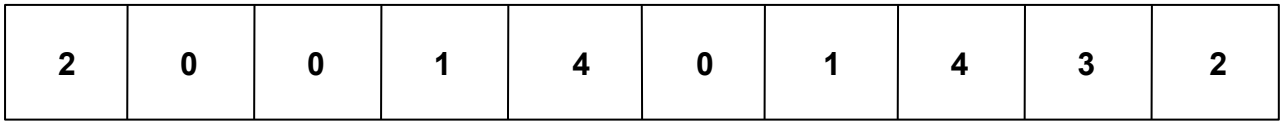
  - Prefix-sum the `numNeighbors` array

```
numNeighbors[i] += numNeighbors[i-1];
```

  - `numNeighbors[i]` now holds the starting index in the Neighbors array of the neighbors of vertex `i`

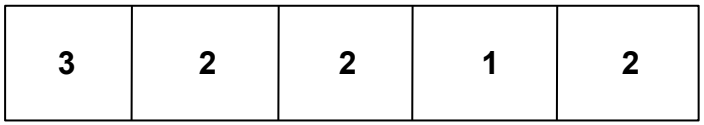
Random Graph Generator Example

Want five vertices, average two out-edges each.

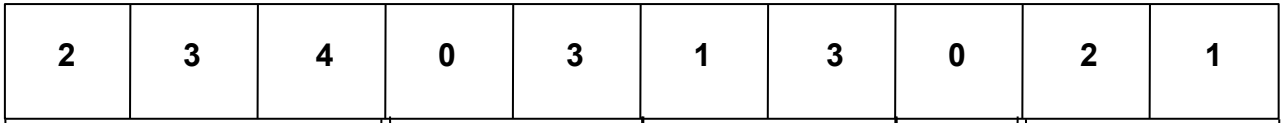



```
randNeighbors(Neighbors, …
```

Histogram:

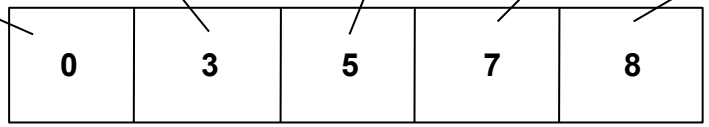



```
numNeighbors[Neighbors[i]]++;
```



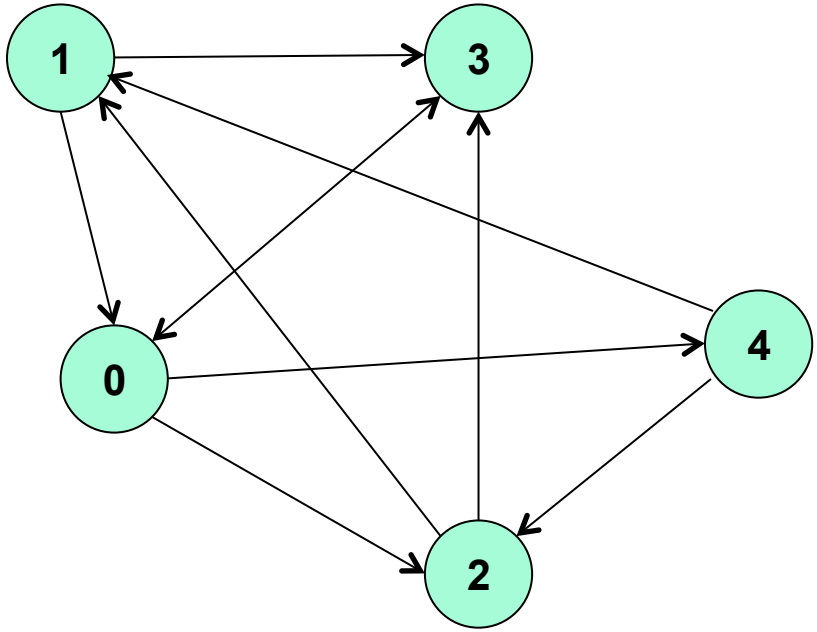

```
randNeighbors(Neighbors, …
```

Prefix sum:



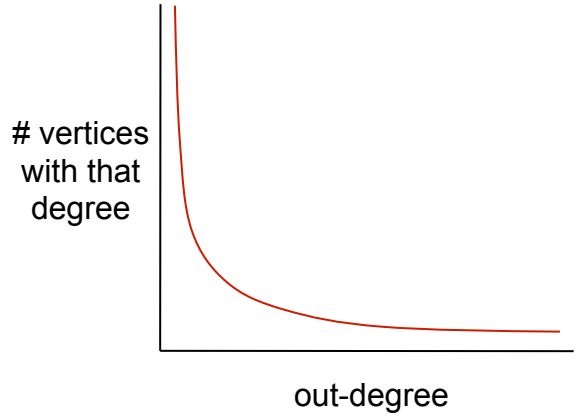

```
numNeighbors[i] += numNeighbors[i-1];
```

Resulting Graph

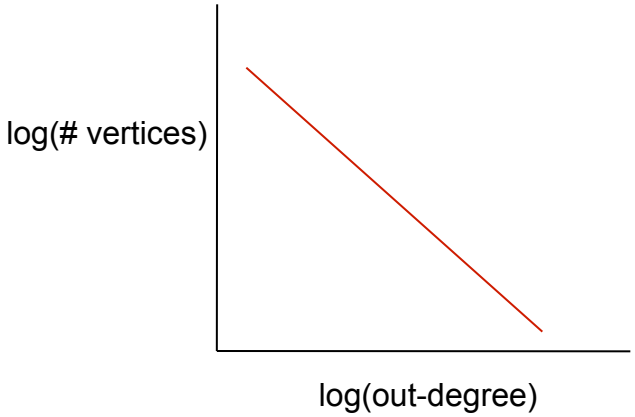


Power Law Graphs

- **Number of vertices with out-degree x = $cx^{-\beta}$**



- **Straight line on a log-log graph, with slope -β**

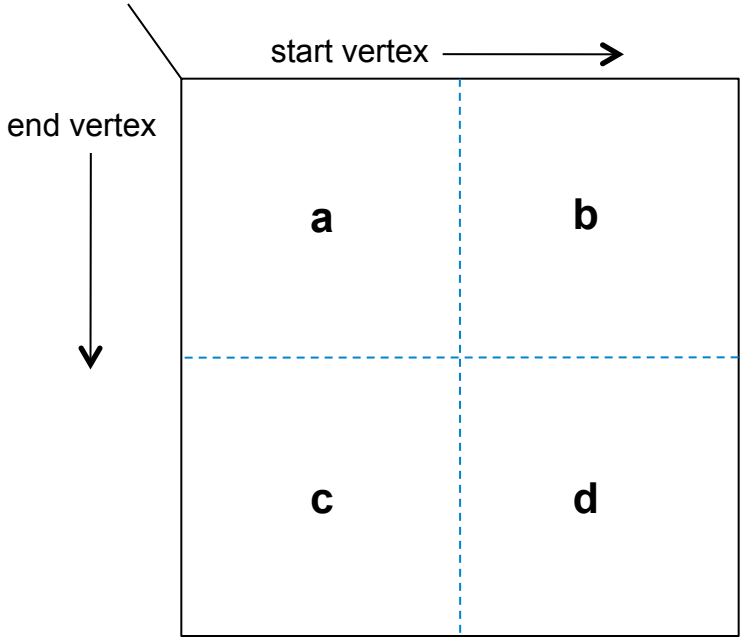


About Power Law Graphs

- A few vertices with huge out-degree ("heavy-tailed" distribution)
- Contain clusters of "communities" (connectivity inside > connectivity outside)
- Found to be much closer fit to many "real world" social networks
  - WWW
  - Internet router graphs
  - Citation graphs
  - Facebook
  - …
- Small diameter but high connectivity ("six degrees of separation")
- AKA "self-similar", "scale-free" graphs

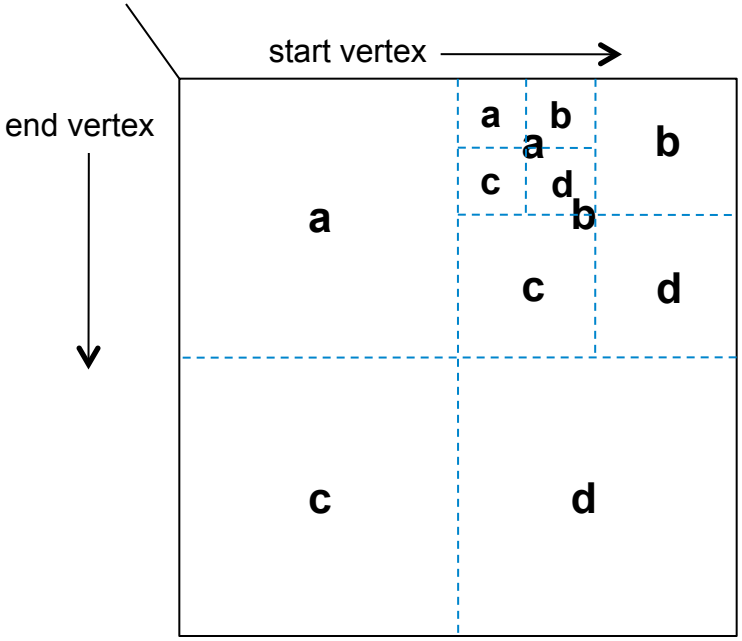
Popular Power Law Graph Generator

- R-MAT = “recursive matrix”
- Invented by Chakrabarti, Yahoo! Research and Falutsos, CMU
- Four probability parameters, a, b, c and d
  - Three, really: d = 1.0 – (a + b + c)
- Divide the graph adjacency matrix into four quadrants



Power Law Graph Generator

- Apply recursively until you reach a single cell
- Our implementation tweaks a, b, c and d ± at most 10% every iteration
- It also uses a hash table to eliminate duplicate edges
- Typical parameters are a=.55, b=.19, c=.19, d=.07
- SSCA #2 Handout 2 point to file instead???



Parallel I/O Using Snapshot/Restore

Avoiding Graph Generation

- Generating synthetic data is often slow – e.g. graph generation
- We try to avoid it, when possible, by using Snapshot-Restore.
- STEPS:
  - What you put in the source code: **SSCA #2 Handout 3**
  - How you initialize for using the parallel file system

In the current configuration, the parallel file system is /mnt/lustre, a Lustre file system
When fully configured, matterhorn will attach to a DVS filesystem

Snapshot/restore will work identically on both systems

In the Source Code

The `#includes` needed, and some file names

```
#include <luc/luc_exported.h>
#include <snapshot/client.h>

const char  VE_FILENAME[] =
     "/scratch/dmizell/mydata/rmat28_vertex_edge_info.data";
const char  SV2_FILENAME[] =
     "/scratch/dmizell/mydata/rmat28_sv2_snapshot.data";
const char  EV2_FILENAME[] =
     "/scratch/dmizell/mydata/rmat28_ev2_snapshot.data";
const char  START_FILENAME[] =
     "/scratch/dmizell/mydata/rmat28_start_snapshot.data";
const char  WEIGHT_FILENAME[] =
     "/scratch/dmizell/mydata/rmat28_weight_snapshot.data";
```

In the Source Code

Initializing the Snapshot library

```
// Initialize Snapshot Library
// The SWORKER_EP environment variable is read at this point.

    if (err = snap_init() != SNAP_ERR_OK)
    {
        fprintf(stderr,
"Failed to initialize snapshot library. Error %d. \n", err);
    }
```

Snapshot/Restore I/O Calls

Writing files

```
/* Memory allocated for the vertex/edge info data structure */
  int *veptr = (int *)malloc(2*sizeof(int));
  veptr[0] = G->numVertices; veptr[1] = G->numEdges;

// All file system workers must be able to access the specified path.
   numbytes = (size_t)(2*sizeof(int));
   err = snap_snapshot ((char *)VE_FILENAME, veptr,
                        numbytes, &snapError);

   numbytes = (size_t)(G->numEdges*sizeof(int));
   err = snap_snapshot ((char *)SV2_FILENAME, G->startVertex,
                        numbytes, &snapError);

//… more file writes…
```

Snapshot/Restore I/O Calls

Reading files

```
// Restore the Graph from disk
   numbytes = (size_t)(2*sizeof(int));
   err = snap_restore ((char *)VE_FILENAME, veptr, numbytes, &snapError);
   G->numVertices = veptr[0]; G->numEdges = veptr[1];
//…
// All file system workers must be able to access the specified path.
   numbytes = (size_t)(G->numEdges*sizeof(int));
   err = snap_restore ((char *)SV2_FILENAME, G->startVertex, numbytes, &snapError);
//… more file reads…
```

Exercise 4

```
/mnt/lustre/Workshop/Exercise4
```

Optional Homework

Dataflow exercise

/mnt/lustre/Workshop/Homework/wavefront

Integer Sort

/mnt/lustre/Workshop/Homework/sort

The End!

Thanks!

End of Day 2

Backup Slides

Making Reductions and Recurrences Run in Parallel

Thanks to Jon Berry, Sandia

Example 1

Summing the absolute values of integers

- First try (compiler didn't parallelize it)

```
int total=0;
for (int i=0; i<n; i++) {
        if (v[i] < 0) {
                total += -v[i];
        } else {
                total += v[i];
        }
}
```

Second Try

Parallelized successfully

```
int total=0;
for (int i=0; i<n; i++) {
    int incr = (v[i] < 0) * -v[i]  + (v[i] >= 0) * v[i];
    total += incr;
}
```

A Conditional Reduction

- This didn't parallelize:

```
int max=0;
for (int i=0; i<n; i++) {
        if (mask[i] && v[i] > max) {
                max = v[i];
        }
}
```

- This did:

```
int max=0;
for (int i=0; i<n; i++) {
        int candidate = mask[i] * v[i];
        if (candidate > max) {
                max = candidate;
        }
}
```