

uRIKA

The Cray XMT
Multithreaded
Programming
Model

Agenda



Why Multithreading?



The XMT Architecture



Programming the XMT



XMT Performance Tools



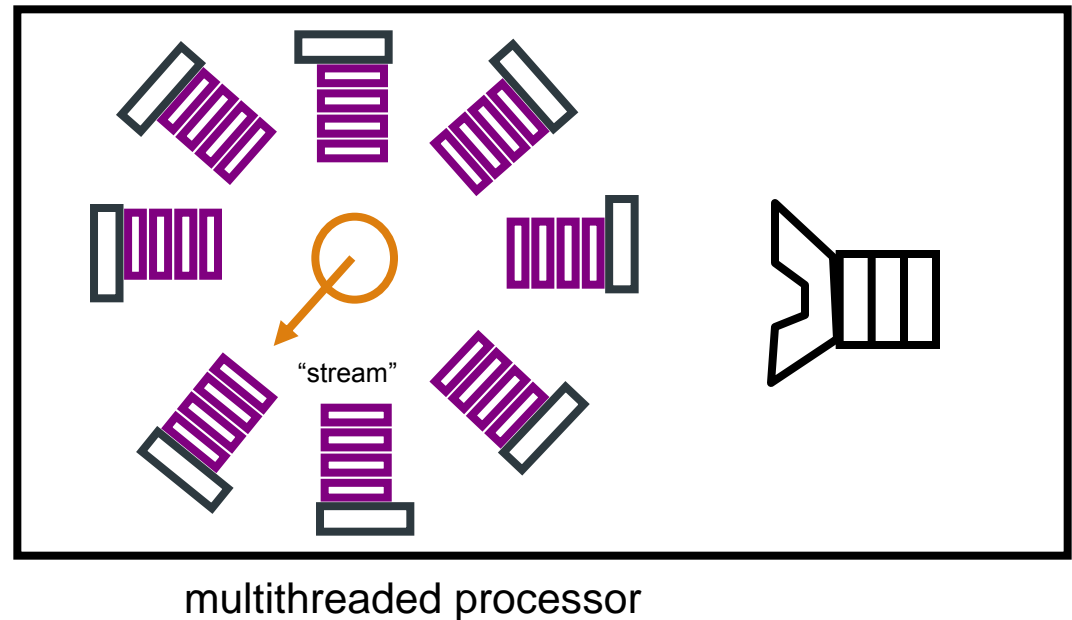
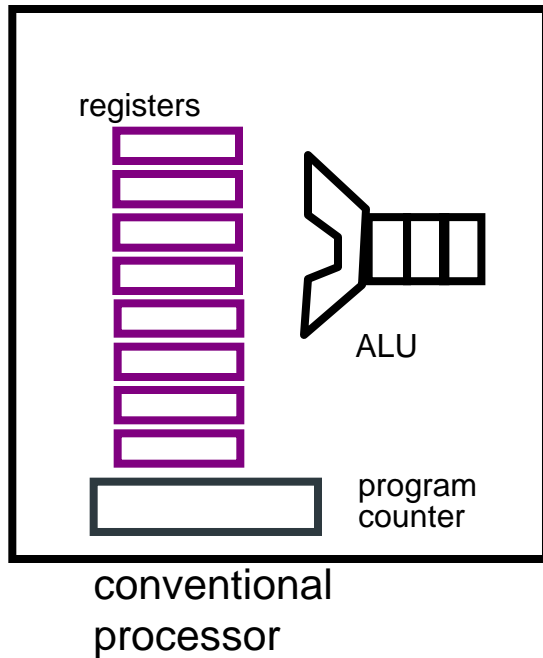
Summary

Why Multithreading?

- **Relative latency to memory continues to increase**
 - Vector processors *amortize* memory latency
 - Cache-based microprocessors *reduce* memory latency
 - Multithreaded processors *tolerate* memory latency
- **Multithreading is most effective when:**
 - Parallelism is abundant
 - Data locality is scarce
- **Large graph problems perform well on this architecture**
 - Semantic databases
 - Big data

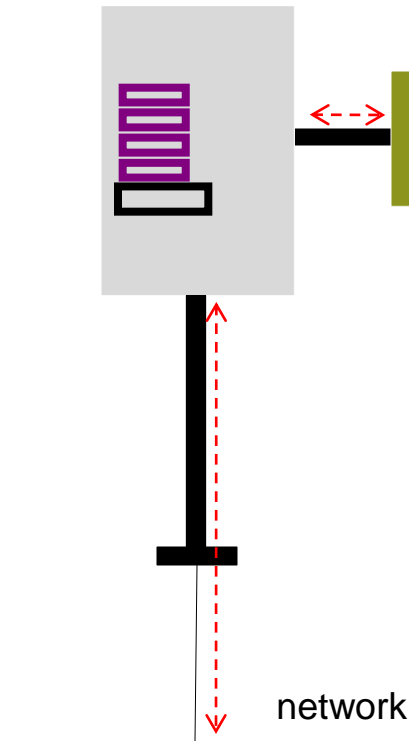
Multithreading Made Simple

- Many threads per processor core; small thread state
- Thread-level context switch at every instruction cycle

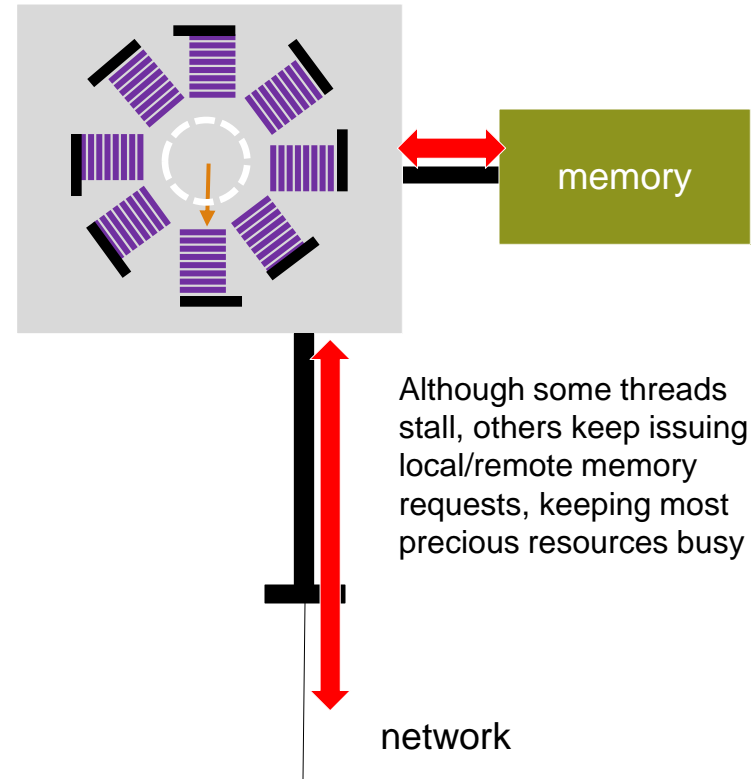


Keeping the Bottlenecks Saturated

- Conventional processor



- Multithreaded processor



The Advantage of Multithreading

- Memory or network latency delaying one thread will not delay processor core with more work to do
- Keeps processors busier but that's less important than the fact that it keeps precious resources – memory and network bandwidth – busier
- Works very well with global shared memory because of its latency tolerance
- Needs fine-grain, cheap synchronization
- Programmer can design algorithms in terms of global shared memory and abundant parallelism, almost like PRAM model

Agenda



Why Multithreading?



The XMT Architecture



Programming the XMT



XMT Performance Tools



Summary

Cray XMT Architecture

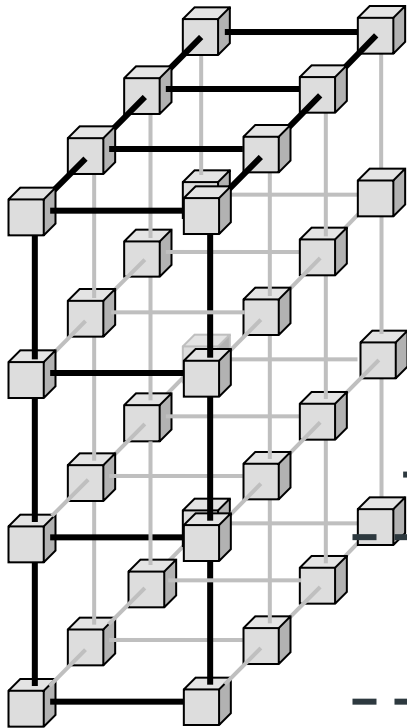
- Heavily multi-threaded processor: 128 hardware threads multiplexed between OS and all applications
 - 16 protection domains (address maps) per processor
 - Multi-threaded architecture tolerates memory latency
 - Data locality not critical for performance
- Scrambled and distributed shared memory to avoid contention
- Lightweight synchronization using full/empty bits on all memory
- Interconnect bisection bandwidth scales with the number of processors
- No hardware interrupts
 - Hardware threads allocated by user via instruction, not OS



XMT System Logical View

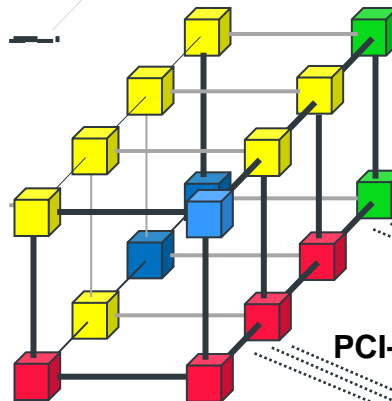
Compute

MTK








Service & IO

Linux



Service Partition

- *Linux OS*
- *Specialized Linux nodes*

-  Login PEs
-  IO Server PEs
-  Network Server PEs
-  FS Metadata Server PEs
-  System Server PEs

Compute Partition

-  MTK (BSD)

PCI-E

10 GigE

Network

PCI-E

Fibre Channel

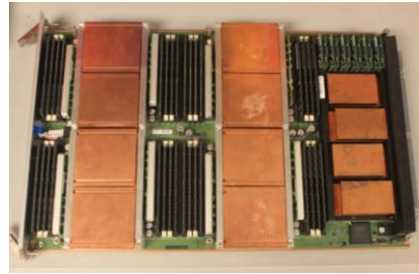
RAID Controllers

Cray XMT2 is Built on the XT5 Infrastructure

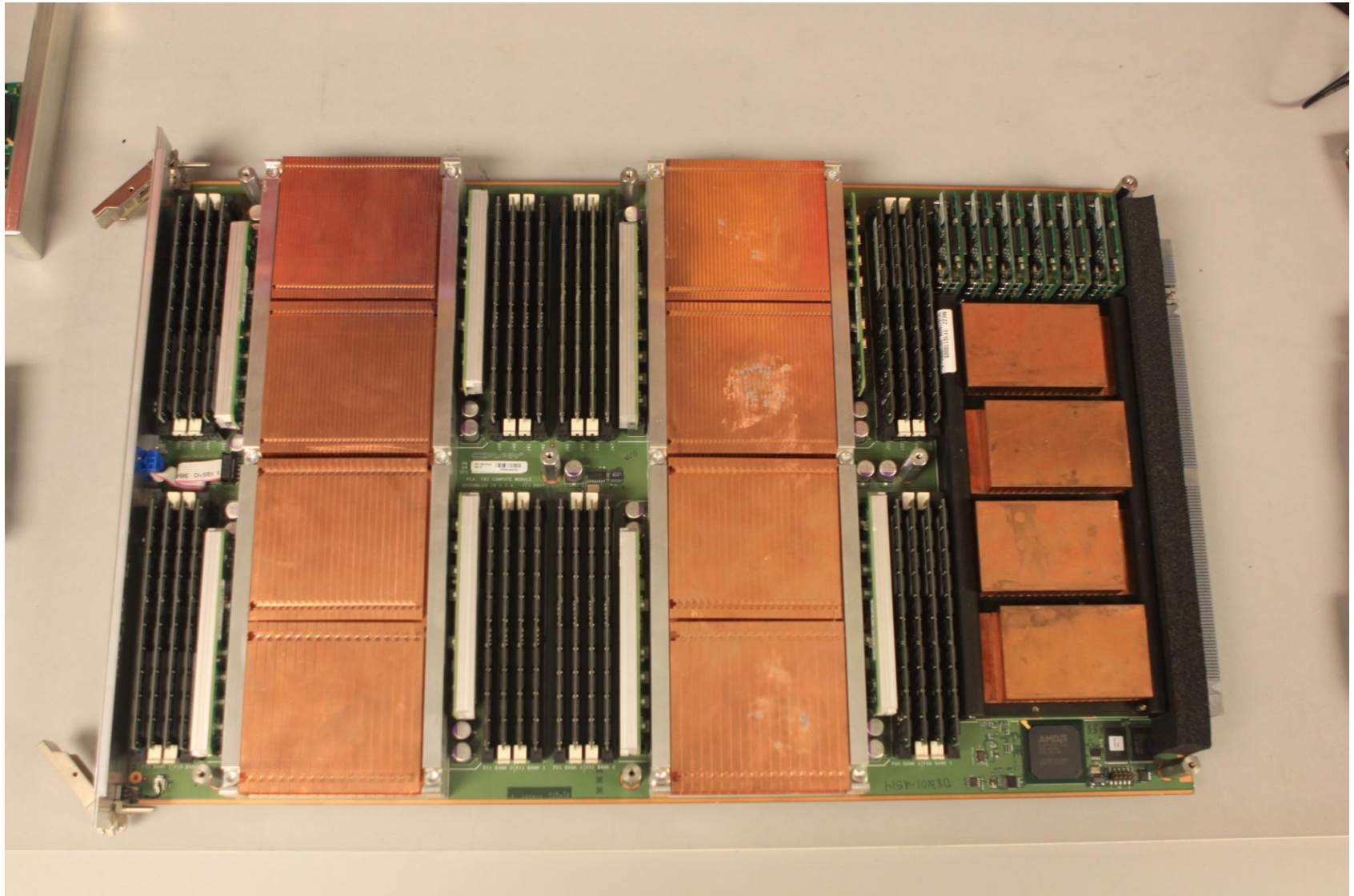
- Uses the same cabinets, boards, scalable interconnect, I/O and storage infrastructure, user environment, and administrative tools...

...just changes the processor

- Cabinet
 - 24 blades per cabinet
 - Vertical airflow with optional liquid assist
- Compute blades
 - 4 Threadstorm processors
 - 16-64 GB per processor
- Cray XT service and I/O subsystem
 - PCIe connections to storage and networks
 - Scalable Lustre global file system
- Cray XT high-speed 3D torus network
- Cray XT power and RAS systems
- Linux based user environment



XMT2 Compute Blade



Agenda



Why Multithreading?



The XMT Architecture



Programming the XMT



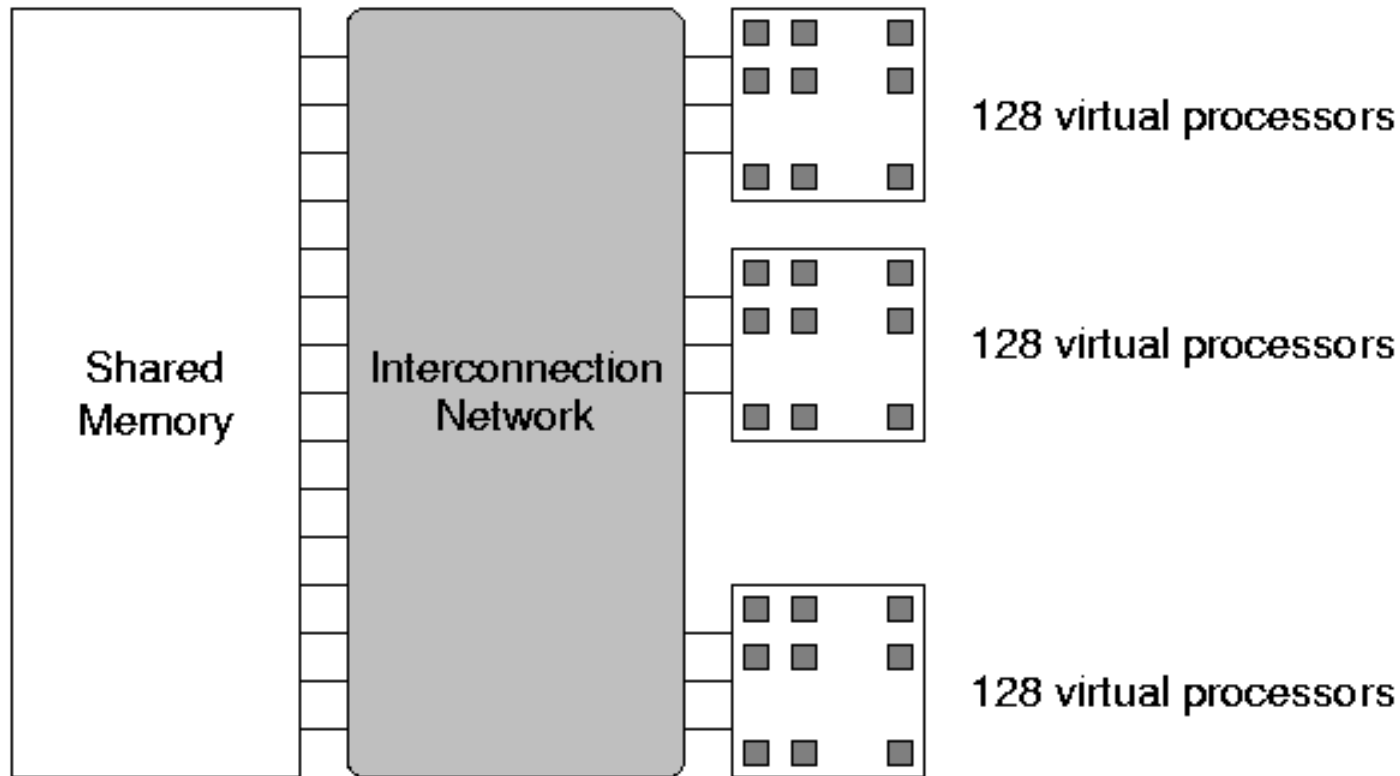
XMT Performance Tools



Summary

XMT Programming Model

- To the programmer, a multiple processor XMT **looks like a single processor**, except that the number of threads is increased.



Bokhari/Sauer 2003

uRiKA Programming Environment

● Automatic Parallelizing C / C++ Compiler

- Compiler runs on Linux login node
- The executable runs on the compute nodes
- Provides automatic parallelism if proper options are included
- Parallelization directives are used to provide “hints” to the compiler

● Login Nodes run standard SuSe (SLES 11) Linux

- All usual Linux languages and applications may be used
- A “hybrid” programming model is possible, and encouraged

● Lightweight user communication (LUC) interface

- Use LUC to build a client/server interface between the front-end and back-end
- Symmetric; RPC-style interface in either direction

● Lustre global file system

- High-performance parallel file system used across Cray line

XMT Compiler

- **C/C++ optimizing compiler**
 - Aggressive automatic parallelization capability
 - Support for various hierarchies of parallelization
 - reductions, linear recurrences
 - Support for atomic memory operations
 - Interprocedural optimization
 - includes capability to inline library functions
- **Incremental recompilation and incremental linking**
- **Tightly integrated with debugging and performance analysis tools**

Using the XMT's C/C++ Compiler

- 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.



Inputs to the Compiler: Pragmas

- The compiler automatically parallelizes loops when it can.
 - The programmer influences the compiler's loop parallelization actions with *pragmas*.
-
- `#pragma mta assert nodep *A`
 - `#pragma mta assert parallel`
 - `#pragma mta loop future`
 - `#pragma mta interleave schedule`
 - `etc.`



Potential Architectural Bottlenecks

- Processor throughput? Never observed to be the bottleneck.
 - Typical processor utilization ~ 30%
- Network bandwidth
 - Especially at larger scale
 - Tunable HW settings in the network, based on the amount of concurrency needed to saturate the bisection BW, in place to avoid over-saturation of the network
 - 128P sized systems and smaller limited to 180 outstanding memory operations per processor
 - For 512P system, this drops to 144 outstanding operations
- Memory bandwidth
 - Minimizing trips to memory is important



Additional Performance Considerations

- Sequential code murders performance.
 - 21 cycles per instruction issue
- XMT memory references are hashed
 - Granularity of 8-word cache lines
- All jobs use all memories in the system
 - Example: “betweenness centrality” graph computation on 16 processors of a 128-processor system: 25% faster than on 16 processors of a 16-processor system
- Exclusive protection domains, but no exclusive ownership of physical hardware resources



Agenda



Why Multithreading?



The XMT Architecture



Programming the XMT



XMT Performance Tools



Summary

Performance tools 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



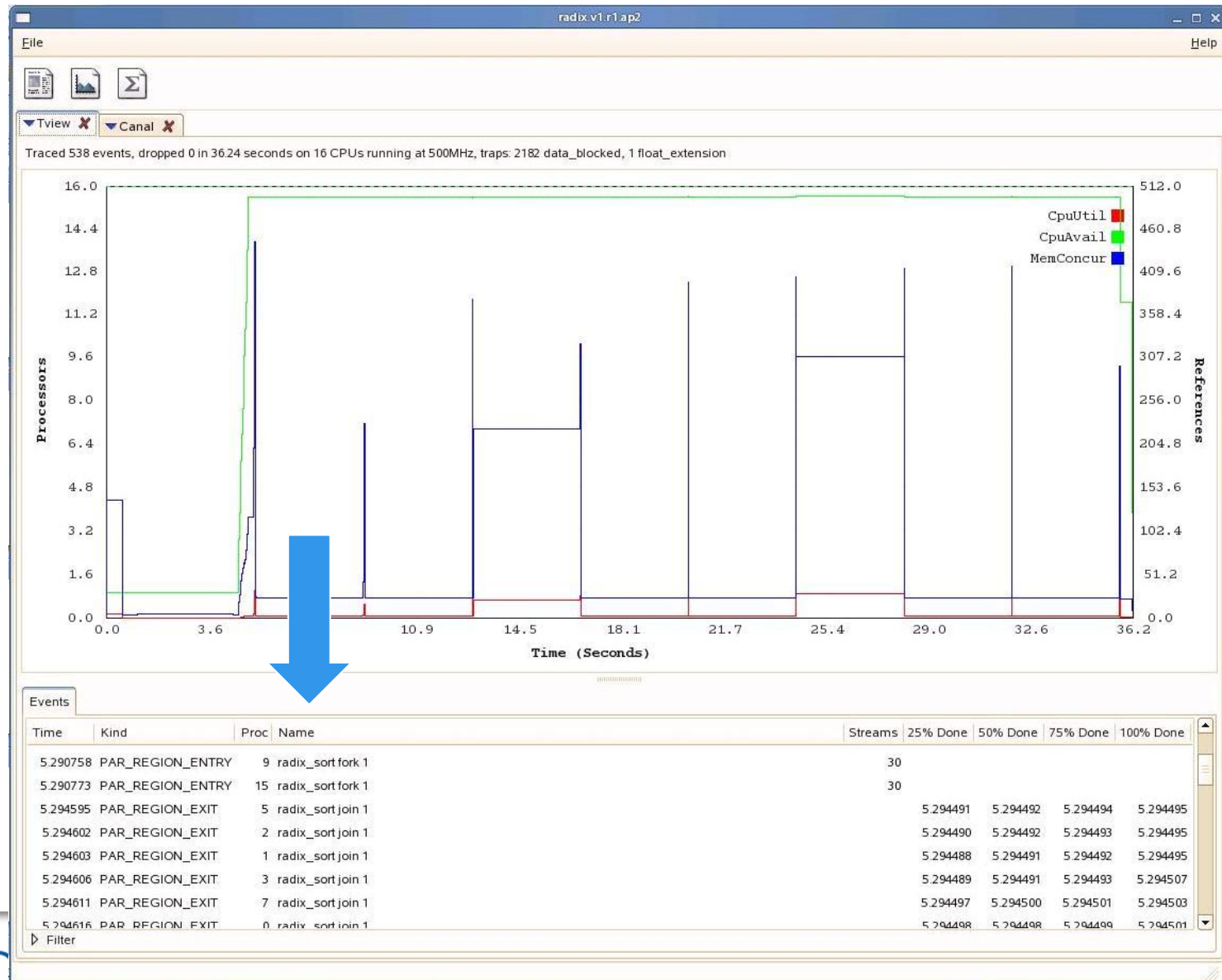
Canal – compiler analysis

- The Canal tool provides feedback from the compiler on whether/how it parallelized loops.
- One section of the Canal output is associated with source code:

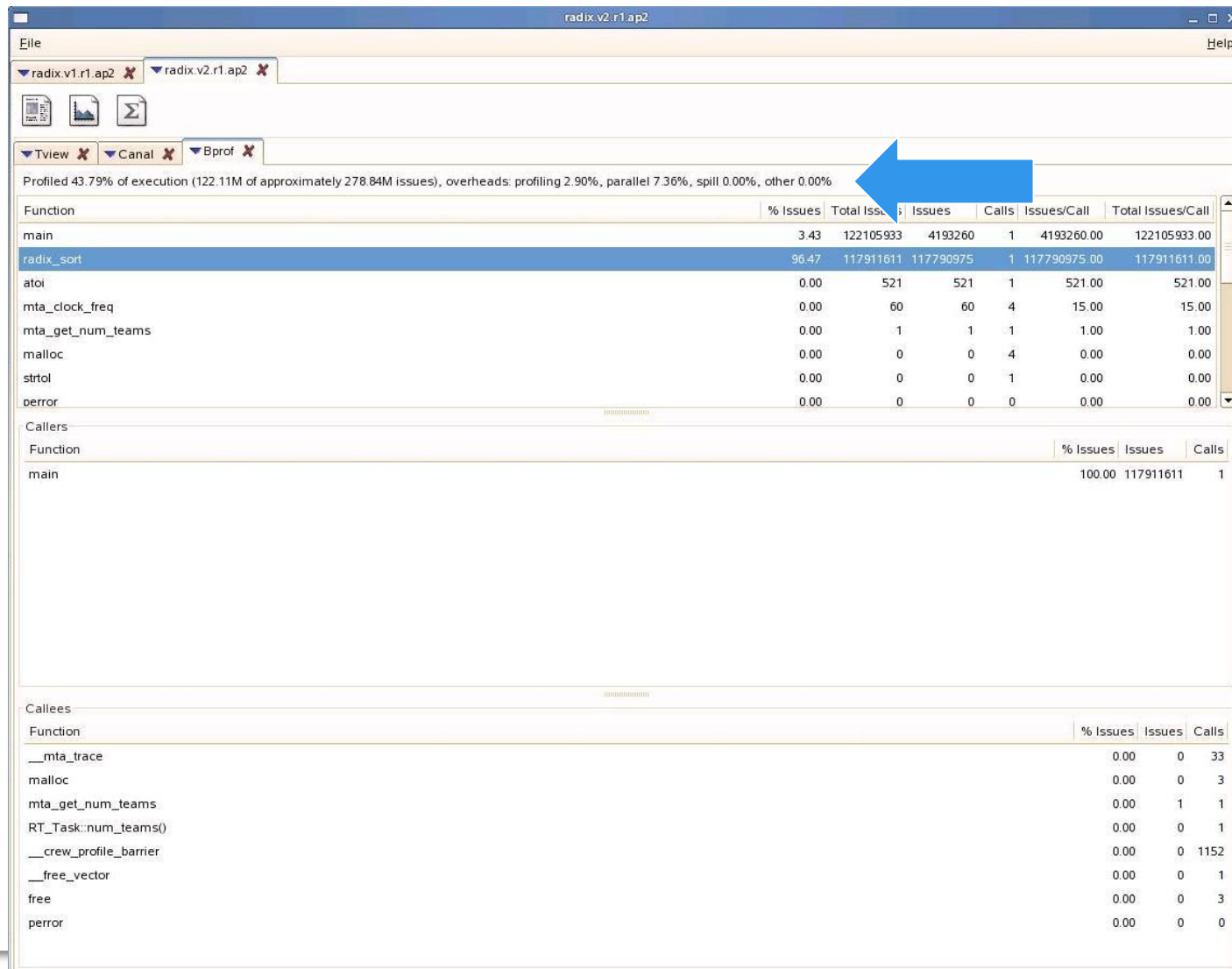
```
2 p      | #pragma mta assert parallel
          | #pragma mta use 100 streams
          | for(int th=0; th<MTA_NUM_STREAMS(); th++) {
          |     unsigned outhead = 0, outtail = 0;
          |     for(;;) {
          |         // grab INBLOCK nodes (& stubs) from the input
3 DX      |         unsigned inhead = int_fetch_add(&newhead, INBLOCK);
          |         // avoid overrun
3 pX      |         unsigned intail = min(inhead + INBLOCK, oldtail);
++ function min inlined
          |
```



Traceview - performance monitor



Bprof – performance profiling



Agenda



Why Multithreading?



The XMT Architecture



Programming the XMT



XMT Performance Tools



Summary

Summary

- **The Multithreaded programming model has superior performance on problems that involve...**
 - High degree of data parallelism
 - Large fraction of remote references
- **The Cray XMT2 is based on proven supercomputer hardware with a custom multithreaded processor**
- **Thinking in parallel and working closely with the compiler lead to best performance**
- **A wide array of tools are available for performance optimization**

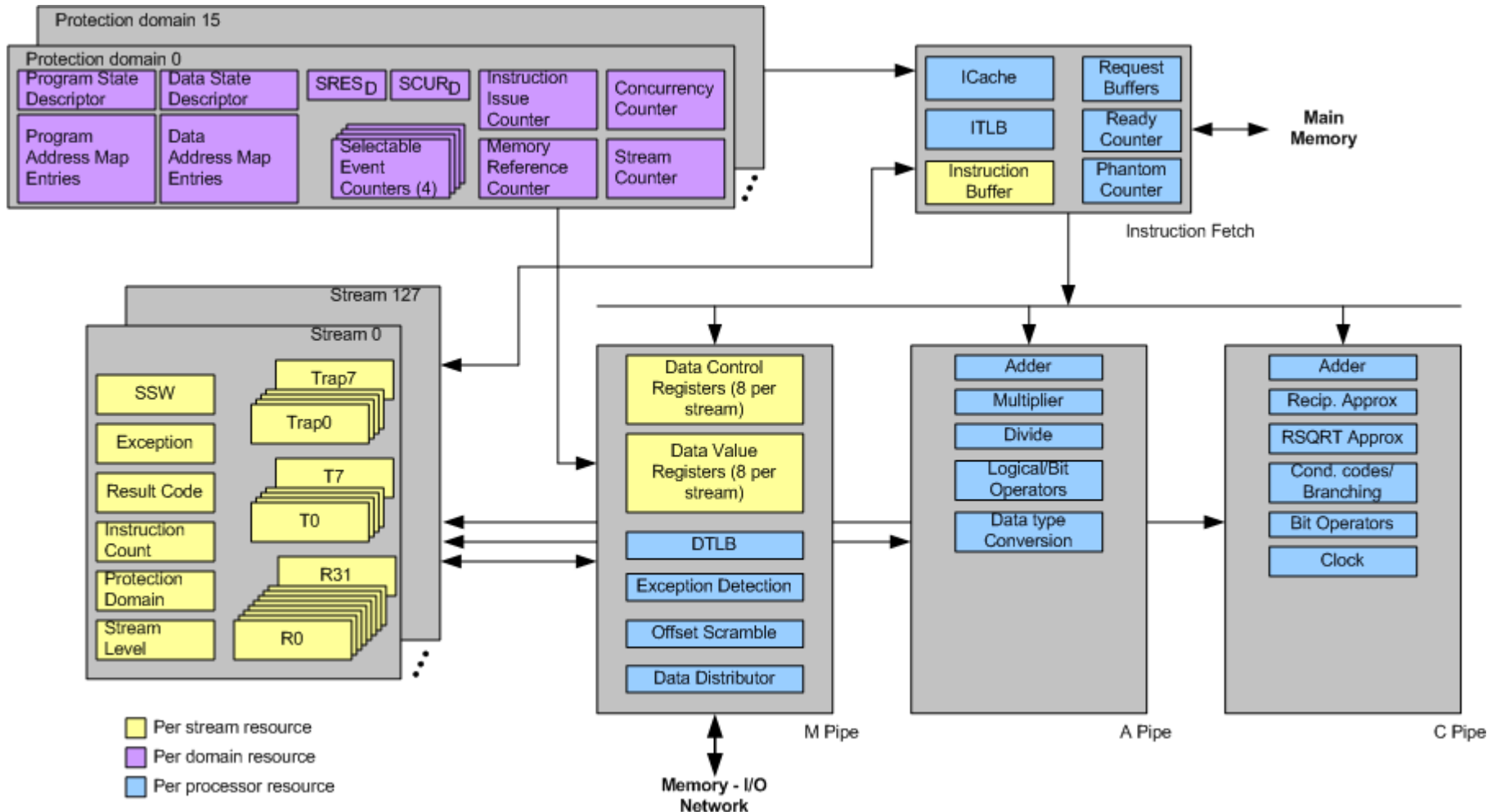


Thank you!

James D. Maltby, Ph.D
jmaltby@yarcdata.com

Backup Slides

XMT's "Threadstorm" CPU Architecture



XMT2 Compute Blade

