A methodical approach for scaling applications on Multi-core MPP Systems

John Levesque & Luiz DeRose
Cray's
Supercomputing Center of Excellence

The steps – 1) Identify Application and Science Worthy Problem

- Formulate the problem
  - The problem identified should make good science sense
    - No publicity stunts that are not of interest
  - It should be a production style problem
    - Weak scaling
      - Finer grid as processors increase
      - Fixed amount of work when processors increase
    - Strong scaling
      - Fixed problem size as processors increase
        - Less and less work for each processor as processors increase

Think Bigger

The steps – 2) Understand the target system Hardware and Software

- Hardware
  - Node Architecture
  - Interconnect
  - Input-Output
- Software
  - Operating System
  - Parallel I/O software
  - Programming Environment
    - Compilers
- Programming Considerations
  - Cache Optimization
  - Vectorization
  - Efficient MPI
  - OpenMP

Multi-core, MPP systems are very similar; however, there are important differences

The steps – 3) Instrument the application

- Instrument the application
  - Run the production case
    - Run long enough that the initialization does not use > 1% of the time
    - Run with normal I/O
  - Use Craypat's APA
    - First gather sampling for line number profile
    - Second gather instrumentation (-g mpi,io)
      - Hardware counters
      - MPI message passing information
      - I/O information

load module
make
pat_build -O apa a.out
Execute
pat_report *.xf
pat_build –O *.apa
Execute

The steps – 4) Examine Results

- Examine Results
  - Is there load imbalance?
    - Yes – fix it first – go to step 5
    - No – you are lucky
  - Is computation > 50% of the runtime
    - Yes – go to step 6
  - Is communication > 50% of the runtime
    - Yes – go to step 7
  - Is I/O > 50% of the runtime
    - Yes – go to step 8

Always fix load imbalance first

The steps – 5) Application is load imbalanced

- What is causing the load imbalance
  - Computation
    - Is decomposition appropriate?
    - Would RANK_REORDER help?
  - Communication
    - Is decomposition appropriate?
    - Would RANK_REORDER help?
    - Are receives pre-posted
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMP-ize large amount of code
- Go back to step 3
    - Re-gather statistics

Need Craypat reports

Is SYNC time due to computation?

The steps – 6) Computation is Major Bottleneck

CRAY
THE SUPERCOMPUTER COMPANY

- What is causing the Bottleneck?
  - Computation
    - Is application Vectorized
      - No – vectorize it
    - What library routines are being used?
  - Memory Bandwidth
    - What is cache utilization?
    - TLB problems?
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMPize large amount of code
- Go back to step 3
  - Re-gather statistics

Need Hardware counters & Compiler listing in hand

The steps – 6) Communication is Major Bottleneck

- What is causing the Bottleneck?
  - Collectives
    - MPI_ALLTOALL
    - MPI_ALLREDUCE
    - MPI_REDUCE
    - MPI_VGATHER/MPI_VSCATTER
  - Point to Point
    - Are receives pre-posted
      - Don't use MPI_SENDRECV
    - What are the message sizes
      - Small – Combine
      - Large – divide and overlap
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMP-ize large amount of code
- Go back to step 3
    - Re-gather statistics

Look at craypat report
MPI message sizes

The steps – 7) I/O is Major Bottleneck

- What type of I/O?
  - One writer – large files
    - Stripe across most OSTs
  - All writers – small files
    - Stripe across one OST
  - MPI-I/O?
    - Try using subset of writers
  - Go back to step 3
    - Re-gather statistics

Look at craypat report on file statistics
Look at read/write sizes

Vectorization

- Stride one memory accesses
- No IF tests
- No subroutine calls
  - Inline
- What is size of loop
- Loop nest
  - Stride one on inside
  - Longest on the inside
- Unroll small loops
- Increase computational intensity
  - CU = (vector flops/number of memory accesses)

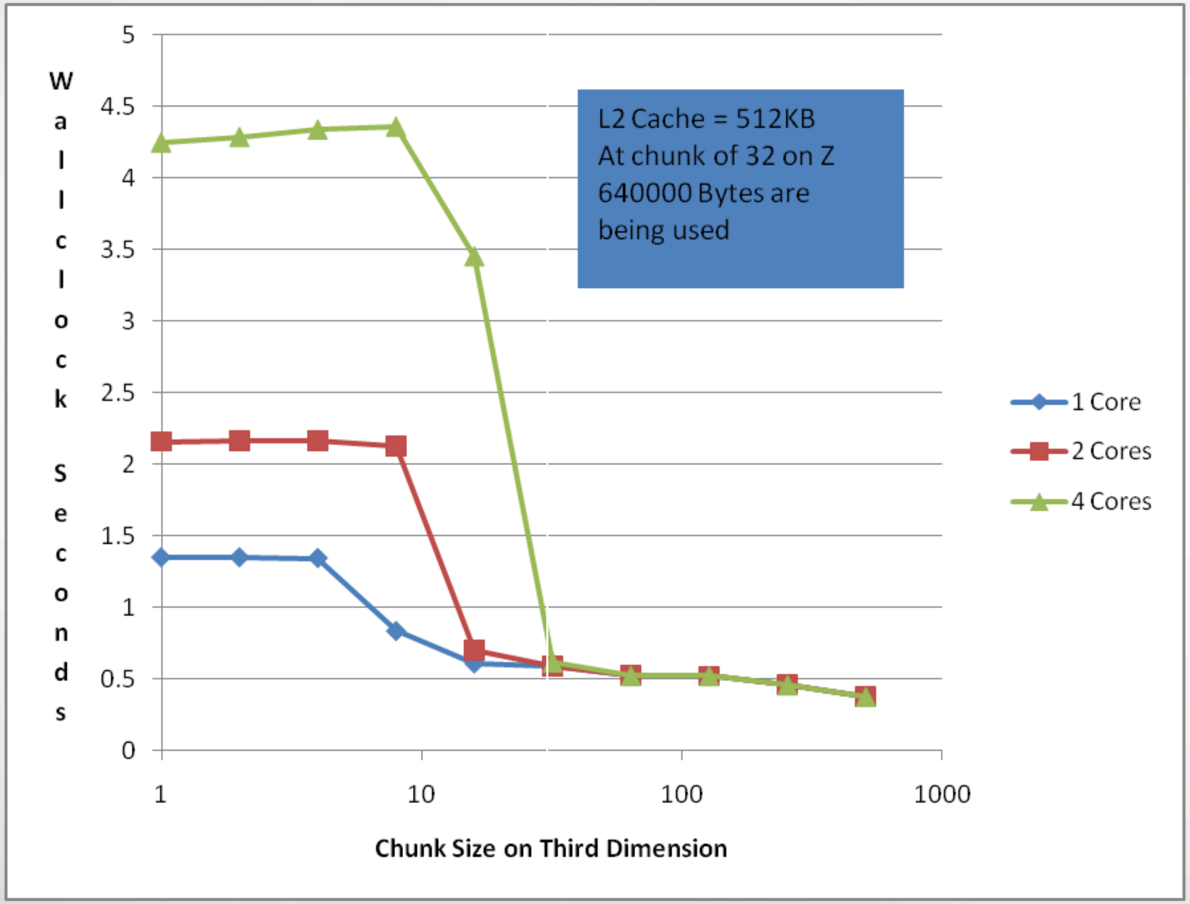
Simple Strip Mining loop

```
 integer, parameter :: nx=100, ny=100, nz=512, nc=100
  real(r4) a(nx,ny,nz),s
!..... initialize array a: a(ix,iy,iz)=ix+(nx*((iy-1)+ny*(iz-1)))
  in=1
  do il=1,10
  call system_clock(count=start_time)
  do ic=1,nc*in
    do iz=1,nz/in
    do iy=1,ny
    do ix=1,nx
      a(ix,iy,iz)=a(ix,iy,iz)*2.0
    end do
    end do
    end do
    do iz=1,nz/in
    do iy=1,ny
    do ix=1,nx
      a(ix,iy,iz)=a(ix,iy,iz)*0.5
    end do
    end do
    end do
  end do
  call system_clock(count=stop_time)
  in=in*2
  end do
  end
```

Storage Analysis

NX	NY	NZ	Ic	Mwords	MB	L1 Refills	L2 Refills	L3 Refills
100	100	512	1	5.12	40.96	625.00	81.92	40.96
100	100	256	2	2.56	20.48	312.50	40.96	20.48
100	100	128	4	1.28	10.24	156.25	20.48	10.24
100	100	64	8	0.64	5.12	78.13	10.24	5.12
100	100	32	16	0.32	2.56	39.06	5.12	2.56
100	100	16	32	0.16	1.28	19.53	2.56	1.28
100	100	8	64	0.08	0.64	9.77	1.28	0.64
100	100	4	128	0.04	0.32	4.88	0.64	0.32
100	100	2	256	0.02	0.16	2.44	0.32	0.16

Running code across 1, 2, and 4 cores



TLB Utilization

- Must be striding in array
  - Reorganize looping structures
- Use large pages

Background: Virtual Memory

- Modern programs operate in “virtual memory”
  - Each program thinks it has all of memory to itself
  - Fixed sized blocks (“pages”) vs variable sized blocks (“segments”)
- Virtual Memory benefits
  - Allow a program that is larger than physical memory to run
    - Programmer does not have to manually create overlays
  - Allow many programs to share limited physical memory
- Virtual Memory problems
  - Each virtual memory reference must be translated into a physical memory reference

Translation Speed

- Translation page table is stored in main memory
  - Each memory access logically takes twice as long – once to find the physical address, once to get the actual data
- Use a hardware cache of least recently used addresses
  - Called a Translation Lookaside Buffer or TLB

Performance Problem: TLB Refills

- AMD dual core opteron: 512 data TLB entries
- Covers 2MB of physical memory
  - OK if program fits (unlikely)
  - Large programs accessing data from all over their virtual memory range can trigger excessive TLB misses (“thrash”)
- One solution: huge pages

CRAY

THE SUPERCOMPUTER COMPANY