Lecture 16

Multi-tier computing
CSE 260 Symposium

- Schedule posted
- Slight change in allotted times
- Solo: 18 minutes
- Teams of 2: 20 minutes
- Teams of 3: 22 minutes
Today’s lecture

• Multi-tier systems
• Latency tolerance
A Crosscutting issue: Hiding latency

- Little’s law [1961]
  - The number of threads must equal the parallelism times the latency
    \[ T = p \times \lambda \]
  - \( p \) and \( \lambda \) are increasing with time
- Difficult to implement
  - Split phase algorithms
  - Partitioning and scheduling
- The state-of-the-art enables but doesn’t support the activity
- Distracts from the focus on the domain science
- Implementation policies entangled with correctness issues
  - Non-robust performance
  - High development costs
Hybrid processing

- Special case of heterogeneous processor
- Two types of processors: general purpose + accelerator
- Accelerator can perform certain tasks more quickly subject to various overhead costs
- Accelerator amplifies relative cost of communication
Motivating application

- Solve Laplace’s equation in 3 dimensions with Dirichlet Boundary conditions
  \[ \Delta \varphi = \rho(x,y,z), \ \varphi=0 \text{ on } \partial \Omega \]
- Building block: iterative solver using Jacobi’s method (7-point stencil)

for \((i,j,k)\) in \(1:N \times 1:N \times 1:N\)
\[ u'[i][j][k] = \frac{(u[i-1][j][k] + u[i+1][j][k] + u[i][j-1][k] + u[i][j+1][k] + u[i][j][k+1] + u[i][j][k-1])}{6.0} \]
Classic message passing implementation

- Decompose domain into sub-regions, one per process
  - Transmit **halo regions** between processes
  - Compute **inner region** after communication completes
- Loop carried dependences impose a strict ordering on communication and computation
Latency tolerant variant

- Only a subset of the domain exhibits loop carried dependences with respect to the halo region
- Subdivide the domain to remove some of the dependences
- We may now sweep the inner region in parallel with communication
- Sweep the annulus after communication finishes
A few implementation details

• Some versions of MPI can realize overlap with MPI_IRecv and MPI_Isend
• If not, then we can use multithreading to handle the overlap
• We let one or more processors (proxy thread(s)) handle communication
A performance model of overlap

• Assumptions

\[ p = \text{number of processors per node} \]

\[ \text{running time} = 1.0 \]

\[ f < 1 = \text{communication time} \]

\[ \text{(i.e. not overlapped)} \]

\[ 1 - f \]

\[ f \]

\[ T = 1.0 \]
Performance

- When we displace computation to make way for the proxy, computation time \textit{increases}.
- Wait on communication drops to zero, ideally.
- When \( f < \frac{p}{(2p-1)} \): improvement is \((1-f)x\left(\frac{p}{(p-1)}\right)^{-1}\)
- Communication bound: improvement is \( \frac{1}{1-f} \)
NPACI Blue Horizon

- Multiple SMP nodes
  - 144 8-way Power3+ “high” nodes
  - 375 MHz CPU
  - 4 GB memory per node
  - 64 KB L1$, 4MB L2$ per processor
  - Caches have a 128 Byte line size

- Differential MPI communication rates
  (peak Ring)
  - 400 MB/sec off-node
  - 500 MB/sec on node
Performance improves with overlap

- Computation time increases with fewer CPUs
- Loss is mitigated by memory system saturation at high levels of parallelism

Discuss the OPT Variant in detail
Where we are so far

• The synchronous (SY) and an overlapped (OV) variants differ in two fundamental ways
  – SY employs a single level of parallel control flow and it exhibits distinct computation and communication phases
  – OV has two levels of control flow and it communicates asynchronously to tolerate latency. Elaborate partitioning required

• Scheduling is messy, and each application has its own overlap structure
Processor Virtualization

- Use the classic non overlapped variant
- Virtualize the processors by overdecomposing
- AMPI [Kalé et al.]
- On an IBM SP3 system, a net slowdown
- When an MPI call blocks, thread yields to another virtual process
- How do we inform the scheduler about ready tasks?
Non-SPMD programming

• Asynchronous task graph model of execution
  – Communication and computation do not execute as distinct phases but are coupled activities
  – Tolerate unpredictable or irregular task and communication latencies

• 2 projects: Thyme and Tarragon
  – Jake Sorensen and Pietro Cicotti
A graph based approach

• Represent the program as a task precedence graph encoding data dependences
• Virtualized tasks: many to each processor
• Background run time services support dataflow execution of the graph
• The graph maintains meta-data to inform the scheduler about runnable tasks

for (i,j,k) in 1:N x 1:N x 1:N
    u[i][j][k] = .....
Graph execution semantics

- Parallelism exists among independent tasks
- Independent tasks may execute concurrently
- A task is **Runnable** when its data dependences have been met
- A task **Suspends** if its **data dependences** are not met
- Computation and data motion are coupled activities
- Scheduler doesn’t affect graph execution semantics
Run time services

- Background services manage graph execution
- The **scheduler** determines which task(s) to run next
- Scheduler and application are only vaguely aware of one another
Observations

- The exact execution order depends on the data dependence structure: communication & computation
- We don’t have to hard code a particular overlap strategy
- We can alter the behavior by changing the data dependences, e.g. disable overlap, or by varying the on-node decomposition geometry
- For other algorithms we can add priorities to force a preferred ordering
- Applies to many scales of granularity (i.e. memory locality, network, etc.)
Results with Thyme

Thunder (LLNL)
4-way, 1.4 GHz Itanium (8 GB)
Quadrics Elan 4

Datastar (SDSC)
8-way, 1.5 GHz IBM
Power4+ (16 GB)
Federation Interconnect

256 processors

Jake Sorensen

Jacobi3D

SUMMA

NAS-FT

BASE: hand-coded MPI w/ BLOCKing communication

IDEAL: same as base, w/ communication DISABLED

OLAP: hand-coded MPI w/ split phase communication

thyme
Jacobi3D

Thunder

25 iterations

DataStar

$4^3 \times 4^3$

VP = 16:1

$(4\times2\times4) \times 4^3$

VP = 8:1
SUMMA

- 2 panels per block
- Used meta data to change LIFO scheduling policy to prioritize older blocks

Thunder

DataStar

Slower matrix multiply-adds
Cell Broadband Engine
Communication

4 rings
Each link: 16B
Runs at 1/2 the clock rate
Up to 16 outstanding requests
204.8 GB/sec

Rambus XDR DRAM interface
  3.2 Gbit/sec/SPE
  25.6 Gb/s aggregate

Courtesy of Mercury Computer Systems, Inc.
The Cell Chip

- 241M transistors
- 235mm$^2$
- 8 + 1 cores
Offloading work to SPEs

Courtesy of Mercury Computer Systems, Inc.
Workflow
Information flow rates

Synergistic Processor Elements for High (FL) ops / Watt

16B/cycle

16B/cycle

16B/cycle (2x)

16B/cycle

16B/cycle

15B/cycle

64-bit Power Architecture w/VMX for
Traditional Computation

Scott B. Baden/CSE 260/Fall 2009
Software managed resources

- Optimize the common case
- In order issue
- Parallelism in VMX ops and across SPEs rather than ILP
- Aligned accesses
- Branch hints
- No cache on SPEs: local memory
- Large register file 128-bit x 128
- Scheduling of SPEs
DMA requests in flight

- 16 outstanding requests per SPU
SIMD arithmetic instructions

\[ b[i] + c[i] \]

16-byte boundaries

R1

\[
\begin{array}{cccc}
  b0 & b1 & b2 & b3 \\
\end{array}
\]

R2

\[
\begin{array}{cccc}
  c0 & c1 & c2 & c3 \\
\end{array}
\]

R3

\[
\begin{array}{cccc}
  b0+ & b1+ & b2+ & b3+ \\
  c0 & c1 & c2 & c3 \\
\end{array}
\]

16-byte boundaries

Courtesy of International Business Machines Corporation.
Unauthorized use not permitted
Branching

• Assume all branches taken
• Optimizations
  – Predication
  – Branch hint
  – Vector merge
Dual Broadband Engine

Infiniband Daughtercard

512 MB XDR DRAM

25.6 GB/s

3.2 GHz Cell Processor

25.6 GB/s each way

Power

BladeCenter Midplane Connector

Power

GbE

GbE

PCI Express x4

20 GB/s each way

2.5 GB/s each way

South-bridge

South-bridge

3.2 GHz Cell Processor

25.6 GB/s

Serial Port

2.5 GB/s each way

20 GB/s each way

Courtesy of Mercury Computer Systems, Inc.
Dual Broadband Engine
Figures from Kistler et al.
Roadrunner

180 compute nodes  12 I/O nodes
288-port IB 4x DDR

17 CUs

12 links per CU to each of 8 switches

Eight 2nd-stage 288-port IB 4X DDR switches

Triblade node
Roadrunner

![Graph 1: Benefits from Host/Accel Parallelism](image1)

![Graph 2: Components of Normalized Runtime](image2)

![Graph 3: Performance by Problem Size on one CU](image3)