CSE 260
Lecture 20
The future
Announcements
What is the purpose of a supercomputer?

• Improve our understanding of scientific and technologically important phenomenon
• Improve the quality of life through technological innovation, simulations, data processing
  – Data Mining
  – Image processing
  – Simulations – financial modeling, weather, biomedical
• Economic benefits
What is the world’s fastest supercomputer?

- Top500 #1, Tianhe–2 @ NUDT (China)
Tianhe -2

- Developed at National University of Defense Technology in Changsha, China
- #1 on Top500.org
- 3.12 Million cores
- 54.9 Pflop/sec peak
- 17.8 MW power (+6MW for cooling)
- 1 PB memory (2^{50} Bytes)
- Kylin Linux
Performance differs across application domains

- Collela’s 7 dwarfs, patterns of communication and computation that persist over time and across implementations
  - Structured grids
    - Panfilov method
  - Dense linear algebra
    - Matrix multiply, Vector-Mtx Mpy Gaussian elimination
  - N-body methods
  - Sparse linear algebra
    - In a sparse matrix, we take advantage of knowledge about the locations of non-zeros, improving some aspect of performance
- Unstructured Grids
- Spectral methods (FFT)
- Monte Carlo
Application-specific knowledge is important

• There currently exists no tool that can convert a serial program into an efficient parallel program
  … for all applications … all of the time… on all hardware

• The more we know about the application…
  … specific problem … math/physics … initial data …
  … context for analyzing the output…
  … the more we can improve productivity

• Issues
  ‣ Data motion and locality
  ‣ Load balancing
  ‣ Serial sections
The top 4 supercomputers

- Top500.org, November 2013
- Collaborative efforts
- Tens to hundreds of millions of $

K Computer
705K
11.3 (10.5) PF/s
12.7 MW
#1 11/11

Titan
561K (Nvidia K20x+Opteron)
27.2 (17.6) PF/s
8.2 MW
#1 11/12

Sequoia
1.58M cores
20.1 (17.2) PF/s
7.9 MW
#1 6/12

Tianhe 2
3.12M cores
54.9 (33.9) PF/s
17.8 MW
#1 since 6/13
Up and beyond to Exascale

- In 1961, President Kennedy mandated a landing on the Moon by the end of the decade
- July 20, 1969 at tranquility base “The Eagle has landed”
- The US Govt set an ambitious schedule to reach $10^{18}$ flops by 2018
- DOE is taking the lead in the US, EU also engaged
- Massive technical challenges
The Challenges to landing “Eagle”

- High levels of parallelism within and across nodes
  - \(10^{18}\) flops using NVIDIA devices @ \(10^{12}\) flops
  - \(10^6\) devices, \(10^{9+}\) threads
- Power consumption: \(\leq 20\) MW nearly there today, with “just” 0.05 ExaFlops
  - Power consumption 1-2nJ/op today \(\rightarrow\) 20pJ @Exa
  - Data storage & access consumes most of the energy
- Ever lengthening communication delays
  - Complicated memory hierarchies
  - Raise amount of computation per unit of communication
  - Hide latency, conserve locality
- Reliability and resilience
  - Blue Gene L’s MTBF measured in days
- Application code complexity; domain specific languages
  - NUMA processors, not fully cache coherent on-chip
  - Mixture of accelerators and conventional cores
Technological trends

- Growth: cores/socket rather than sockets
- Memory/core is shrinking
- Complicated software-managed parallel memory hierarchy
- Cost of communication increasing relative to computation

Intel Sandybridge, anandtech.com
Trend: data motion costs are rising

- Increase amount of computation performed per unit of communication
  - Conserve locality, “communication avoiding”
- Hide latency
- Many threads

<table>
<thead>
<tr>
<th>Year</th>
<th>Processor Improvement</th>
<th>Memory Improvement</th>
<th>Bandwidth Improvement</th>
<th>Latency Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giga</td>
<td>Peta</td>
<td>Tera</td>
<td>Exa??</td>
<td></td>
</tr>
</tbody>
</table>
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
Motivating application

- Solve Laplace’s equation in 3 dimensions with Dirichlet Boundary conditions
  \[ \Delta \varphi = \rho(x,y,z), \quad \varphi = 0 \text{ on } \partial \Omega \]

- Building block: iterative solver using Jacobi’s method (7-point stencil)

```plaintext
for (i,j,k) in 1:N x 1:N x 1:N
    u'[i][j][k] = (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
```

Scott B. Baden /CSE 260/ Winter 2014
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
Message Passing Interface

MPI_Init(); MPI_Comm_rank();MPI_Comm_size();
Data initialization
MPI_Send/MPI_Isend
MPI_Recv/MPI_Irecv
Computations
MPI_Finalize();
A few implementation details

- Some installations of MPI cannot realize overlap with MPI_IRecv and MPI_Isend
- We can use multithreading to handle the overlap
- We let one or more processors (proxy thread(s)) handle communication

S. Fink, PhD thesis, UCSD, 1998
A performance model of overlap

- Assumptions
  \( p = \text{number of processors per node} \)
  \( \text{running time} = 1.0 \)
  \( f < 1 = \text{communication time} \)
  (i.e. not overlapped)

\[ T = 1.0 \]
Performance

- When we displace computation to make way for the proxy, computation time *increases*
- Wait on communication drops to zero, ideally
- When $f < p/(2p-1)$: improvement is $(1-f)(p/(p-1))^{-1}$
- Communication bound: improvement is $1/(1-f)$
Processor Virtualization

- Virtualize the processors by overdecomposing
- AMPI [Kalé et al.]
- When an MPI call blocks, thread yields to another virtual process
- How do we inform the scheduler about ready tasks?
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.)
Tarragon - Non-SPMD, Graph Driven Execution

- Pietro Cicotti [Ph.D., 2011]
- Automatically tolerate communication delays via a Task Precedence Graph
  - Vertices = computation
  - Edges = dependences

- Inspired by Dataflow and Actors
  - Parallelism ~ independent tasks
  - Task completion ⇔ Data Motion

- Asynchronous task graph model of execution
  - Tasks run according to availability of the data
  - Graph execution semantics independent of the schedule
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
- Background services manage graph execution
- The **scheduler** determines which task(s) to run next
- Scheduler and application are only vaguely aware of one another
- Scheduler doesn’t affect graph execution semantics
Bamboo

- How to leverage Tarragon in existing code?
  - Deterministic procedure for translating MPI code to Tarragon, using dynamic and static information about MPI call sites
- Bamboo: custom, domain specific-translator built using ROSE
  - Tan Nguyen (UCSD)
  - Daniel Quinlan (LLNL), Eric Bylaska (PNNL)
  - John Weare (UCSD)
Bamboo Transformations

• **Outlining**
  - TaskGraph definition: fill various Tarragon methods with input source code blocks

• **MPI Translation**: capture MPI calls and generate calls to Tarragon
  - Some MPI calls removed, e.g. Barrier(), Wait()
  - Conservative static analysis to determine task dependencies

• **Code reordering**: reorder certain code to accommodate Tarragon semantics
## Example with directives

<table>
<thead>
<tr>
<th></th>
<th>Send block</th>
<th>Recv block</th>
<th>Compute block (optional in most cases)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma bamboo olap</td>
<td>#pragma bamboo send</td>
<td>#pragma bamboo receive</td>
<td>#pragma bamboo compute</td>
</tr>
</tbody>
</table>

### Example:

- **Pre-amble**
  ```c
  1 MPI_Init(&argc, &argv);
  2 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  3 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  4 Compute process ID of left/right/up/down processors;
  5 Allocate U, V, SendGhostcells, RecvGhostcells;
  6 #pragma bamboo olap
  7 for it = 1 to num_iterations {
    8     #pragma bamboo send{
    9         Pack boundary values to message buffer;
   10         MPI_Isend(SendGhostcells) to left/right/up/down;
   11     }
   12     #pragma bamboo receive{
   13         MPI_Recv(RecvGhostcells) from left/right/up/down;
   14         Unpack incoming data into ghost cells;
   15     }
   16     MPI_Waitall();
   17   for j = 1 to N/Nprocs_Y - 2 {
   18       for i = 1 to N/Nprocs_X - 2 {
   19           V[j,i] = (U[j-1,i] + U[j+1,i] + U[j,i-1] + U[j,i+1])/4;
   20           swap(U,V);
   21       }
   22   free U, V, SendGhostcells, RecvGhostcells;
   23   MPI_Finalize();
  ```

- **Post-amble**

Scott B. Baden / CSE 260 / Winter 2014
The generated code

```cpp
class user_define_task {
    void vinit() {...}
    void vexecute() {...}
    void vinject(Message* msg) {...}
}

int main(int argc, char** argv) {
    MPI_Init();
    Tarragon::initialize();
    Graph graph = new Graph(task);
    graph->accept(processor_layout);
    Tarragon::graph_init(graph);
    Tarragon::graph_execute(graph);
    Tarragon::finalize();
    MPI_Finalize();
}
```

Psuedocode of a Tarragon program
3D Jacobi Results

Cores on Hopper @ NERSC (Cray XE-6)
Strong Scaling, size = 3072^3

Scott B. Baden /CSE 260/ Winter 2014
Inside a supercomputer
Blue Gene

- IBM-US Dept of Energy collaboration
- Low power, high performance interconnect
- 3rd Generation: Blue Gene/Q
- Sequoia at LLNL, #3 on top 500
  - 17.2 Pflops/s (Linpack), 7.9 MW
  - 1.6M cores/93,304 compute nodes, 16 GB mem/node
  - 96 racks, 3,000 square feet
  - 64-bit PowerPC (A2 core)
  - Weighs about the same as 30 elephants
- https://computing.llnl.gov/tutorials/bgq
Hierarchical Packaging

1. Chip
   16 cores

2. Module
   Single Chip

3. Compute Card
   One single chip module,
   16 GB DDR3 Memory

4. Node Card
   32 Compute Cards,
   Optical Modules, Link Chips, Torus

5a. Midplane
   16 Node Cards

5b. I/O Drawer
   8 I/O Cards
   8 PCIe Gen2 slots

6. Rack
   2 Midplanes
   1, 2 or 4 I/O Drawers

7. System
   96 racks, 20PF/s

SC '10, via IBM

Scott B. Baden /CSE 260/ Winter 2014
Blue Gene/Q Interconnect

- 5D toroidal mesh (end around)
  - Can scale to > 2M cores
  - 2 GB/sec bidirectional bandwidth (raw)
    on all 10 links, 1.8 GB available to the user
  - 5D nearest neighbor exchange
    ~1.8GB/s/link, 98% efficiency
  - Hardware latency ranges from 80 ns to 3μs

- Collective network
  - Global Barrier, Allreduce, Prefix Sum
  - Floating point reductions at ~95% of peak
Die photograph

- 16 (user) + 1 (OS) cores
- 18th redundant core
- Each core 4-way multithreaded@1.6 GHz
- Quad wide (@Double) FPU: SIMD
- 42.6 GB/s DDR3 B/W
- Network routing integrated on-chip (5D torus)
- Compare with layout of the Cray 1 (1976)

Private L1 (16K/16K)
Shared L2$: 32MB
Scott B. Baden /CSE 260/ Winter 2014
Architectural directions
Hybrid processing

- Two types of processors: general purpose + accelerator
  - AMD fusion: 4 x86 cores + hundreds of Radeon GPU cores
- Accelerator can perform certain tasks more quickly than the conventional cores
- Accelerator amplifies relative cost of communication, latency hiding important, but not sufficient
Memory structure

- Only partial cache coherence on-chip
- NUMA

http://www.ector.ac.uk/cse/documentation/Phase2b/#arch
Software

• We need to think hard about hiding an explosion of details
• Parallel programming languages
• Domain specific languages
• Embedded domain specific languages
• Autotuning