Scheduling for Numerical Linear Algebra Library at Scale

Jack Dongarra
INNOVATIVE COMPUTING LABORATORY
University of Tennessee
Oak Ridge National Laboratory
University of Manchester
Something’s Happening Here...

- In the “old days” it was: each year processors would become faster.
- Today the clock speed is fixed or getting slower.
- Things are still doubling every 18-24 months.
- Moore’s Law reinterpreted.
  - Number of cores double every 18-24 months.

From K. Olukotun, L. Hammond, H. Sutter, and B. Smith.
500 Fastest Computers Over Time

- **NEC Earth Simulator**
  - 478 TF/s (2007)
- **IBM BlueGene/L**
  - 6.96 PF/s (2007)
- **SUM**
  - 100 Tflop/s (2007)
- **IBM ASCI White**
  - 5.9 TF/s (2007)
- **Intel ASCI Red**
  - 1.17 TF/s (2007)
- **Fujitsu ‘NWT’**
  - 59.7 GF/s (2007)
- **N=1**
  - 1 Tflop/s (2007)

**N=500**

- **1 Gflop/s (2007)**
- **10 Gflop/s (2007)**
- **100 Gflop/s (2007)**
- **1 Tflop/s (2007)**
- **10 Tflop/s (2007)**
- **100 Tflop/s (2007)**
- **1 Pflop/s (2007)**
- **10 Pflop/s (2007)**
- **100 Pflop/s (2007)**

**1 Gflop/s (1993)**

**100 Gflop/s (1993)**

**1 Tflop/s (1993)**

**10 Tflop/s (1993)**

**1 Pflop/s (1993)**

**10 Pflop/s (1993)**

**100 Pflop/s (1993)**
Accelerated Roadrunner
Hybrid Design (2 kinds of chips & 3 kinds of cores)

“Connected Unit” cluster
144 quad-socket dual-core nodes
(138 w/ 4 dual-Cell blades)
InfiniBand interconnects

In aggregate:
8,640 dual-core Opterons + 16,560 eDP Cell chips
76 TeraFlops Opteron + ~1.7 PetaFlops Cell
172,800 cores

15 CU clusters

2nd stage InfiniBand interconnect
(15x18 links to 8 switches)

RR-14

1152 AMD cores / cluster each core with a Cell processor
Performance Development & Projections

- 10^6 Threads
- 10^9 Threads

1 Gflop/s (O(1) Thread)
1 Tflop/s (O(10^3) Threads)
1 Pflop/s (O(10^6) Threads)
1 Eflop/s (O(10^9) Threads)
Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems.

- Using 4 MW today, going to 15 MW before year end.
- By 2012 could be using more than 50 MW!!

Cost estimates based on $0.07 per KwH.

Includes both DOE and NSF systems.
What’s Next?

Different Classes of Chips
- Home
- Games / Graphics
- Business
- Scientific

The question is not whether this will happen but whether we are ready.
What Will a Petascale System Looks Like?

<table>
<thead>
<tr>
<th>Possible Petascale System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. # of cores per nodes</td>
</tr>
<tr>
<td>2. Performance per nodes</td>
</tr>
<tr>
<td>3. Number of nodes</td>
</tr>
<tr>
<td>4. Latency inter-nodes</td>
</tr>
<tr>
<td>5. Bandwidth inter-nodes</td>
</tr>
<tr>
<td>6. Memory per nodes</td>
</tr>
</tbody>
</table>

- **In general would like high...**
  1. performance per node
  2. performance per node
  5. bandwidth inter-nodes
  6. memory per nodes

- **Algorithms for multicore and need for latency avoiding algorithms**
  1. Number of cores per node
  2. performance per node
  4. Latency inter-nodes

- **Issues involving fault tolerance**
  - Motivation in:
    1. Number of cores per node
    3. number of nodes
Major Changes to Software

- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- Numerical libraries for example will change
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this
LAPACK and ScaLAPACK

About 1 million lines of code
Coding for an Abstract Multicore

Parallel software for multicores should have two characteristics:

- **Fine granularity:**
  - High level of parallelism is needed
  - Cores will probably be associated with relatively small local memories. This requires splitting an operation into tasks that operate on small portions of data in order to reduce bus traffic and improve data locality.

- **Asynchronicity:**
  - As the degree of thread level parallelism grows and granularity of the operations becomes smaller, the presence of synchronization points in a parallel execution seriously affects the efficiency of an algorithm.
ManyCore - Parallelism for the Masses

- We are looking at the following concepts in designing the next numerical library implementation
  - Dynamic Data Driven Execution
  - Self Adapting
  - Block Data Layout
  - Mixed Precision in the Algorithm
  - Exploit Hybrid Architectures
  - Fault Tolerant Methods
Steps in the LAPACK LU

- **DGETF2** (Factor a panel)
- **DLSWP** (Backward swap)
- **DLSWP** (Forward swap)
- **DTRSM** (Triangular solve)
- **DGEMM** (Matrix multiply)

LAPACK

BLAS
LU Timing Profile (4 core system)

Threads – no lookahead

Time for each component

Bulk Sync Phases

- DGETF2
- DLASWP(L)
- DLASWP(R)
- DTRSM
- DGEMM
Adaptive Lookahead - Dynamic

Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress();
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress();
        case END:
            for()
                dlaswp();
            return;
    }

Reorganizing algorithms to use this approach
Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

Column-Major
Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

- **Column-Major**
- **Blocked**
LU -- 8-way dual Opteron -- MKL-9.1

![Diagram showing performance comparison with problem size and Gflop/s]
Cholesky on the CELL

1 CELL (8 SPEs)
- 186 Gflop/s
- 91 % peak
- 97 % SGEMM peak

2 CELLs (16 SPEs)
- 365 Gflop/s
- 89 % peak
- 95 % SGEMM peak

Single precision results on the Cell
If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system
Each Node or Core Will Have

- BIN 1
  - some dependencies satisfied
  - waiting for all dependencies

- BIN 2
  - all dependencies satisfied
  - some data delivered
  - waiting for all data

- BIN 3
  - all data delivered
  - waiting for execution
DAG and Runtime

Bin 1: Waiting for dependencies to be satisfied
Bin 2: All dependencies satisfied, waiting for data
Bin 3: dependencies and data available; ready to execute

- Execute task in Bin 3
  - Task with all data and dependencies satisfied
  - After execution report to children done and dependencies satisfied and send data
  - Steal task if none
- Check Bin 1 to see if new dependencies satisfied for tasks
  - If new dependency satisfied update and post receive of data
  - If all dependencies and data available satisfied move to Bin 2
- Check Bin 2 to see data arrival
  - Check for data arrival; If all data available move to Bin 3
- If needed place new task from my part of the DAG into Bin 1
DAG and Scheduling

- DAG is dynamically generated and implicit
- Everything designed for distributed memory systems
- Runtime system on each node or core

- Run time
- Bin 1
  - Exec a task that’s ready
  - Notify children of completion
  - Send data to children
  - If no work do work stealing
- Bin 2
  - See if new dependences are satisfied
  - If so move task to Bin 3
- Bin 3
  - See if new data has arrived
Task Life-Cycle

dep

dep

dep

req data

req data

req data

execute

notify

notify

notify
Looking Into a Number of Things

• DAG must be dynamic
  ▪ Some of the algorithms are iterative i.e. eigenvalue problem

• Parameterized Task Graph
  ▪ Cosnard and Jeannot

• DAG has to be handled in a distributed fashion
Some Questions

- What’s the best way to represent the DAG?
- What’s the best approach to dynamically generating the DAG?
- What run time system should we use?
  - We will probably build something that we would target to the underlying system’s RTS.
- What about work stealing?
  - Can we do better than nearest neighbor work stealing?
- What does the program look like?
  - Experimented with Cilk, Charm++, UPC, Intel Threads
  - I would like to reuse as much of the existing software as possible
Collaborators / Support

Alfredo Buttari, ENS/INRIA
Julien Langou, U Colorado, Denver
Julie Langou, UTK
Piotr Luszczek, MathWorks
Jakub Kurzak, UTK
Stan Tomov, UTK