Initial Results on Heterogeneity and Energy for Linear Algebra Algorithms

Grey Ballard, James Demmel, Andrew Gearhart

CS 294
September 30, 2011

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung.
Previous Models of Computation

- Machine models assume “fast” and “slow” types of memory access
- We wish to asymptotically minimize “slow” traffic
  - goal of the models is to capture the design space of likely best algorithms...while leaving specific parameter selection to autotuners
Previous Models of Computation

- Machine models assume “fast” and “slow” types of memory access
- We wish to asymptotically minimize “slow” traffic
  - goal of the models is to capture the design space of likely best algorithms...while leaving specific parameter selection to autotuners

**Sequential**: Single processor separated from memory via a small cache
  - fast = cache, slow = DRAM

**Distributed**: A group of processors connected on a network
  - fast = local, slow = remote
Hetero is already here!!!! AHHHH!!!!

- Non-uniform memory access (NUMA), GPU/CPU computation, frequency scaling, etc.

So...we’re gonna have to deal with it...

Image cropped from Edvard Munch’s "Scream"
Sources of heterogeneity

- Heterogeneity in space
  - Performance parameter nonuniformity fixed in hardware
  - Examples: NUMA, hardware production effects, cores with different ISAs?

- Heterogeneity in time
  - Performance parameters may vary during the execution of an algorithm
  - Examples: frequency scaling, OS context switching, bus contention, DRAM bank state?

- Controllable vs. uncontrollable
  - Some sources of heterogeneity can be accessed to tune performance, and others must be ignored or mitigated
Sources of heterogeneity

- **Heterogeneity in space**
  - Performance parameter nonuniformity fixed in hardware
  - Examples: NUMA, hardware production effects, cores with different ISAs?

- **Heterogeneity in time**
  - Performance parameters may vary during the execution of an algorithm
  - Examples: frequency scaling, OS context switching, bus contention, DRAM bank state?
Sources of heterogeneity

- **Heterogeneity in space**
  - Performance parameter nonuniformity fixed in hardware
  - Examples: NUMA, hardware production effects, cores with different ISAs?

- **Heterogeneity in time**
  - Performance parameters may vary during the execution of an algorithm
  - Examples: frequency scaling, OS context switching, bus contention, DRAM bank state?

- **Controllable vs. uncontrollable**
  - Some sources of heterogeneity can be accessed to tune performance, and others must be ignored or mitigated
Heterogeneous Model

- We can build a basic model by assuming that each processor has an independent link to a shared global memory and that system parameters are constant.

- Processor $i$ has parameters:
  - $M_i$: local memory size (words)
  - $\gamma_i$: processing rate (sec/flop)
  - $\alpha_i$: latency (sec/msg)
  - $\beta_i$: inverse bandwidth (sec/word)

- e.g., multicore CPU + GPU, 8 ARM cores + 2 Nehalem cores, ...
But wait. Is this model realistic? Should each processor really have its own independent link to global memory?
But wait. Is this model realistic? Should each processor really have its own independent link to global memory?

Uh...maybe. Bandwidth and latency benchmarking show a more complex story...
Machine Model: Considerations

- FSB-based machine:

PCIe bandwidth w/ pinned memory

Xeon E5405 (FSB-based, 8 cores) and GTX280 GPU

Transfer Size (doubles)
Machine Model: Considerations

- NUMA machine:

PCle bandwidth w/ pinned memory

Xeon E5530 (NUMA, 8 cores) and Tesla C2050 GPU
Machine Model: Considerations

PCIe Latency (dirac)

each trial 10000 ping-pong

Trial

Time (sec)

Busy
Busy (avg)
Idle
For the purposes of these results...

- Unless otherwise stated, we will be considering the independent links model
For the purposes of these results...

- Unless otherwise stated, we will be considering the independent links model

- (due to simplicity and for some cleaner communication bounds)
We model a parallel program’s total runtime $T$ by

$$T(\{F_i, W_i, L_i\}) = \max_{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i W_i + \alpha_i L_i \right\}$$

$\gamma_i = \text{sec/flop}$  \quad $\beta_i = \text{sec/word}$  \quad $\alpha_i = \text{sec/msg}$

$F_i = \text{flops}$  \quad $W_i = \text{words}$  \quad $L_i = \text{messages}$
We model a parallel program’s total runtime $T$ by

$$T(\{F_i, W_i, L_i\}) = \max_{1 \leq i \leq P} \{ \gamma_i F_i + \beta_i W_i + \alpha_i L_i \}$$

- $\gamma_i = \text{sec/flop}$
- $\beta_i = \text{sec/word}$
- $\alpha_i = \text{sec/msg}$
- $F_i = \text{flops}$
- $W_i = \text{words}$
- $L_i = \text{messages}$

Previous bounds for sequential model:
- assuming (3 nested loops) linear algebra computations

$$W_i \geq \frac{F_i}{8\sqrt{M_i}} \quad L_i \geq \frac{F_i}{8M_i^{3/2}}$$
Applying these bounds to each processor we obtain a runtime lower bound for any partition of $G$ flops to $P$ processors:

$$T \geq \min \max \sum_{F_i=G}^{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i \frac{F_i}{8\sqrt{M_i}} + \alpha_i \frac{F_i}{8M_i^{3/2}} \right\}$$
Heterogeneous Runtime Lower Bound

Applying these bounds to each processor we obtain a runtime lower bound for any partition of $G$ flops to $P$ processors:

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i \frac{F_i}{8\sqrt{M_i}} + \alpha_i \frac{F_i}{8M_i^{3/2}} \right\}$$

which can be simplified by solving the linear program:

$$T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \{ \delta_i F_i \} = \frac{G}{\sum \frac{1}{\delta_i}}$$

where

$$\delta_i = \gamma_i + \frac{\beta_i}{8\sqrt{M_i}} + \frac{\alpha_i}{8M_i^{3/2}}$$
Heterogeneous Runtime Lower Bound

Applying these bounds to each processor we obtain a runtime lower bound for any partition of $G$ flops to $P$ processors:

$$T \geq \min \sum_{F_i=G} \max_{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i \frac{F_i}{8\sqrt{M_i}} + \alpha_i \frac{F_i}{8M_i^{3/2}} \right\}$$

which can be simplified by solving the linear program:

$$T \geq \min \sum_{F_i=G} \max_{1 \leq i \leq P} \left\{ \delta_i F_i \right\} = \frac{G}{\sum \frac{1}{\delta_i}}$$

where

$$\delta_i = \gamma_i + \frac{\beta_i}{8\sqrt{M_i}} + \frac{\alpha_i}{8M_i^{3/2}}$$

- this bound is attainable for sufficiently large dense matrix multiplication
- the solution to LP gives optimal partition of flops (insight for algorithm)

$$F_i = \frac{\frac{1}{\delta_i} G}{\sum \frac{1}{\delta_j}}$$
More general (and possibly tighter) lower bound for sequential model:

\[ W_i \geq \max \left\{ I_i + O_i, \frac{F_i}{8\sqrt{M_i}} \right\}, L_i \geq \max \left\{ \left\lceil \frac{I_i + O_i}{M_i} \right\rceil, \frac{F_i}{8M_i^{3/2}} \right\} \]

where \( I_i \) and \( O_i \) are the number of input and output words for processor \( i \), respectively.

If we assume that the serial lower bounds apply for all processors, we obtain a general lower bound for program runtime:

**Theorem (General Heterogeneous Lower Bound)**

For \( P \) processors and \( G \) total flops to execute, then

\[ T \geq \min_{\sum F_i = G} \max_{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i \max \left\{ I_i + O_i, \frac{F_i}{8\sqrt{M_i}} \right\} + \alpha_i \max \left\{ \left\lceil \frac{I_i + O_i}{M_i} \right\rceil, \frac{F_i}{8M_i^{3/2}} \right\} \right\} . \]
Heterogeneous Lower Bounds: matrix-vector operations

- In the case of matrix-vector operations $I + O = O(n^2) = G$, so another valid lower bound is

$$T(\{F_i\}) \geq \min \sum_{F_i=G} \max_{1 \leq i \leq P} \left\{ \gamma_i F_i + \beta_i (cF_i) + \alpha_i \left( \frac{cF_i}{M_i} \right) \right\}$$

with $c$ constant.

- Again we solve the linear program (with different constants) to simplify and obtain the optimal partition of flops for matrix-vector operations:

$$T \geq \frac{G}{\sum_j \frac{1}{\xi_j}} \quad \text{and} \quad F_i = \frac{1}{\xi_i} G$$

where

$$\xi_i = \gamma_i + c\beta_i + \frac{c\alpha_i}{M_i}$$
Runtime model with shared bus

Independent link model

\[ T = \max_i \{\gamma_i F_i + \beta_i W_i + \alpha_i L_i\} \]

\[ T \geq \min \max_{\{F_i\}} \delta_i F_i \]
Runtime model with shared bus

Independent link model

\[ T = \max \left\{ \gamma_i F_i + \beta_i W_i + \alpha_i L_i \right\} \]
\[ T \geq \min \max \{ \delta_i F_i \} \]

Shared bus model

\[ T = \beta \sum_i W_i + \sum_i \alpha_i L_i + \max \{ \gamma_i F_i \} \]
\[ T \geq \min \left\{ \beta \sum_i \frac{F_i}{8\sqrt{M_i}} + \sum_i \frac{\alpha_i F_i}{8M^{3/2}} + \max \{ \gamma_i F_i \} \right\} \]

Still a linear program!
are the lower bounds tight?
- yes: matrix-vector multiplication

\[ T \geq \frac{G}{\sum_j \frac{1}{\xi_j}} \quad \text{for} \quad \xi_j = \gamma_i + c\beta_i + \frac{c\alpha_i}{M_i} \]

- yes: matrix-matrix multiplication

\[ T \geq \frac{G}{\sum_j \frac{1}{\delta_j}} \quad \text{for} \quad \delta_i = \gamma_i + \frac{\beta_i}{8\sqrt{M_i}} + \frac{\alpha_i}{8M_i^{3/2}} \]
Algorithm Outline

1. Determine parameters $\alpha_i, \beta_j, \gamma_i, M_i$ and use to calculate each $F_i/G = \frac{1}{\xi_i} \sum_j \frac{1}{\xi_j}$

2. Split the input matrix (stored row-major) according to the values of $F_i/G$

3. Each processor uses an efficient matrix-vector multiplication routine to calculate its assigned work

4. Merge results into output vector in global memory
Heterogeneous Algorithm: Matrix Multiplication

Our algorithm is based on the square recursive algorithm (8 subproblems)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \cdot \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix} = \begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\]
Our algorithm is based on the square recursive algorithm (8 subproblems)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\cdot
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
=
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\]
Heterogeneous Algorithm: Matrix Multiplication

Our algorithm is based on the square recursive algorithm (8 subproblems)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22} \\
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22} \\
\end{bmatrix}
= 
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22} \\
\end{bmatrix}
\]

Algorithm Overview

- Use static scheduling based on measured machine parameters
- Use flop distribution given by linear program from lower bound
  - load balances given optimal communication costs

\[F_i / \delta = \sum_j 1/\delta_j\] converts to octal to determine how many subproblems to assign to processor \(i\) at each level of recursion

minimizes per processor bandwidth cost

minimizes per processor latency cost
Heterogeneous Algorithm: Matrix Multiplication

Our algorithm is based on the square recursive algorithm (8 subproblems)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix} =
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\]

Algorithm Overview

- Use static scheduling based on measured machine parameters
- Use flop distribution given by linear program from lower bound
  - load balances given optimal communication costs
- Assign as many large subproblems to processors as possible
  - convert \( F_i/G = \frac{1}{\delta_i} \sum_j \frac{1}{\delta_j} \) to octal to determine how many subproblems to assign to processor \( i \) at each level of recursion
  - minimizes per processor bandwidth cost
- Use block-recursive data structure so that subproblems are contiguous
  - minimizes per processor latency cost
### Heterogeneous Algorithms: Matrix-matrix multiplication

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>.25</td>
<td>.25</td>
<td>.40625</td>
<td>.09375</td>
</tr>
<tr>
<td>Octal</td>
<td>.20</td>
<td>.20</td>
<td>.32</td>
<td>.06</td>
</tr>
</tbody>
</table>
Heterogeneous Algorithms : Matrix-matrix multiplication

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>.25</td>
<td>.25</td>
<td>.40625</td>
<td>.09375</td>
</tr>
<tr>
<td>Octal</td>
<td>.20</td>
<td>.20</td>
<td>.32</td>
<td>.06</td>
</tr>
</tbody>
</table>

Recursion 1

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Recursion 2

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
Heterogeneous Algorithms: Matrix-matrix multiplication

Recursion 1

<table>
<thead>
<tr>
<th>Decimal</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>.25</td>
<td>.25</td>
<td>.40625</td>
<td>.09375</td>
<td></td>
</tr>
<tr>
<td>Octal</td>
<td>.20</td>
<td>.20</td>
<td>.32</td>
<td>.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursion 1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Recursion 2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Heterogeneous Algorithms: Matrix-matrix multiplication

Recursion 1

P1

A_{11} B_{11} A_{21} B_{11}

A_{12} B_{21} A_{22} B_{21}

A_{11} B_{12} A_{21} B_{12}

A_{12} B_{22} A_{22} B_{22}

Recursion 2

P1 P2 P3 P4

Decimal .25 .25 .40625 .09375

Octal .20 .20 .32 .06

Recursion 1 2 2 3 0

Recursion 2 0 0 2 6

P2

P3

P4
Heterogeneous Algorithms: Matrix-matrix multiplication

**Require:** Matrices $A, B \in \mathbb{R}^{n \times n}$ are stored in block-recursive order, $n$ is a power of two

1: Measure $\alpha_i, \beta_j, \gamma_i, M_i$ for each $1 \leq i \leq P$
2: for $i = 1$ to $P$ do
3: Calculate $F_i$ where $G = n^3$
4: Set $k_i$ to be largest integer s.t. $3(n/2^{k_i})^2 \geq M_i$
5: Round $F_i/G$ to the $k_i^{th}$ octal digit: $0.d_1^{(i)} d_2^{(i)} \cdots d_{k_i}^{(i)}$
6: end for
7: Initialize $S = \{A \cdot B\}$ as set of problems
8: for $j = 1$ to max $k_i$ do
9: Subdivide all problems in $S$ into 8 subproblems
10: for $i = 1$ to $p$ do
11: Assign $d_j^{(i)}$ problems to proc$_i$ and remove from $S$
12: end for
13: end for
14: for all proc$_i$ parallel do
15: Compute assigned subproblems using recursive matrix multiplication
16: end for

**Ensure:** Matrix $C = AB$, stored in block-recursive order
complication: we have to lower bound total runtime instead of words/messages moved because flops are no longer balanced

complication: what’s the right machine model?

cache oblivious algorithms are useful to navigate the different $M_i$’s
  similar to sequential hierarchical model

static scheduling ignores heterogeneity in time
  what if $\alpha_i, \beta_i, \gamma_i, M_i$ change?
Energy and Heterogeneity

- Does taking advantage of heterogeneity allow for energy savings without significant performance cost?

- Considering heterogeneity in time may allow for us “to have our cake and eat it too” for bandwidth bound algorithms!
For our purposes, we care about the total energy consumed by a machine while computing an algorithm.

So, we don’t mind small periods of high power (J/s) unlike the architecture guys.

At the moment, we aren’t considering OS effects on energy (energy consumed by process switching).
Emerging architectures allow for fairly precise energy counting (Sandy Bridge and variants).

The following plots use on-board energy counters on Sandy Bridge (accuracy supported by a wall-power meter).

Currently, the counters measure energy of the entire chip and just the cores...which is a little limiting.

I have a small library for people to play with these, if desired!
Energy and Heterogeneity: Frequency scaling

With compute-bound algorithms, Higher frequency = More performance
Some problems don’t depend on frequency!!
Energy and Heterogeneity: Scaling benefits

over 2X improvement in J/flop at lower clock frequencies...
Energy and Heterogeneity: Thread layout

- Intel’s OpenMP environment allows a nice interface to pin threads to specific cores
- We take advantage of this capability (and Hyperthreading) to further improve efficiency
Energy and Heterogeneity: Thread layout

- Intel’s OpenMP environment allows a nice interface to pin threads to specific cores
- We take advantage of this capability (and Hyperthreading) to further improve efficiency

**Scatter**

<table>
<thead>
<tr>
<th>PC0</th>
<th>PC1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Compact**

<table>
<thead>
<tr>
<th>PC0</th>
<th>PC1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

PC = “physical core”

2 = hardware thread context assigned with thread #2
Energy and Heterogeneity: Thread layout

Effect of thread layout upon energy

sandy bridge (4 threads, hdgemv, N=22000)

More energy benefits from using only a subset of cores...
Energy and Heterogeneity: Thread layout

And once more, we have a tuning parameter to consider...
Energy and Heterogeneity: Precision

gemv single vs. double precision energy

Sandy Bridge, 4 threads, N=22000, scatter layout

Really due to race to halt, but nice to show empirically...
Energy and Heterogeneity: Precision

gemm single vs. double precision energy

Sandy Bridge, 4 threads, N=10000, scatter layout
Open Questions

- Is our model really the best thing to use for algorithm development?

- When can we simply ignore a processor? (i.e. cheaper to do small problems only on CPU, as opposed to CPU/GPU split)

- How big do problems have to be before we start to see benefit from a CA approach?

- What would an energy-optimal algorithm look like on a heterogeneous machine? How does this relate to the CA paradigm?
Future Work

- Extension of bounds to Strassen-like algorithms

- Algorithms for more complicated algorithms: LU and QR
  - Must consider flops/byte mapping and the critical path of the algorithm

- Can a dynamic scheduling/work stealing approach provide more flexibility? (handling heterogeneity in time as well as space)

- Explore the benefit of frequency scaling for sparse linear algebra...
Block Recursive Layout

Each square submatrix below is stored contiguously in memory: