The Landscape of Parallel Computing Research
Multi-core architectures and programming

Jens Wetzl
May 9, 2012
Memory Wall

+ Energy Wall
+ ILP Wall

= Brick Wall

Source: http://www.dailyhaha.com/
The Berkeley View

- “The Landscape of Parallel Computing Research”
- Report published by a multidisciplinary group at Berkeley
- Based on 7 questions:

1. What are the applications?
2. What are common kernels of the applications?
3. What are the hardware building blocks?
4. How to connect them?
5. How to describe applications and kernels?
6. How to program the hardware?
7. How to measure success?

Tension between Embedded & Server Computing

Evaluation:
What I’ll be talking about

Motivation

The Seven Thirteen Dwarves
  The Seven Dwarves
  The Thirteen Dwarves
  Why bother?

Programming Model

Compilers

Hardware
  Communication
  Transactional Memory
The Seven/Thirteen Dwarves

The Seven Dwarves
The Thirteen Dwarves
Why bother?
The Seven Dwarves

Phil Colella: 7 numerical methods important for science and engineering

Dense Linear Algebra
Sparse Linear Algebra
Spectral Methods
N-Body Methods
Structured Grids
Unstructured Grids
Monte Carlo

The Seven Thirteen Dwarves

- **Dwarf / Motif:**
  - Equivalence class for algorithms
  - Members have similar computation and communication patterns
  - At a high level of abstraction

- 13 dwarves: expansion to all algorithmic methods, not just numerical

- Goal: capture all common algorithms with a small number of dwarves
The Thirteen Dwarves

1. Dense Linear Algebra
2. Sparse Linear Algebra
3. Spectral Methods  
   e.g. FFT
4. N-Body Methods  
   e.g. Particle simulations
5. Structured Grids
6. Unstructured Grids
7. MapReduce  
   e.g. Monte Carlo
8. Combinational Logic  
   e.g. Hashing
9. Graph Traversal
10. Dynamic Programming
11. Backtracking and  
    Branch&Bound
12. Graphical Models  
    e.g. Hidden Markov models
13. Finite State Machines
Why bother?

**Evaluation:**
- Benchmark for new architectures and programming models
- Good performance for all dwarves $\Rightarrow$
  good performance for future applications

**Reusability:**
- Experts can provide efficient parametrized versions
- Non-experts can use them as building blocks for parallel applications

**Analysis:**
- Algorithms within a category share characteristics
- Scaling behavior, bottlenecks, ...
Why bother?

■ Evaluation:
  - Benchmark for new architectures and programming models
  - Good performance for all dwarves ⇒
    good performance for future applications

■ Reusability:
  - Experts can provide efficient parametrized versions
  - Non-experts can use them as building blocks for parallel applications

■ Analysis:
  - Algorithms within a category share characteristics
  - Scaling behavior, bottlenecks, ...
Why bother?

■ Evaluation:
  □ Benchmark for new architectures and programming models
  □ Good performance for all dwarves ⇒ good performance for future applications

■ Reusability:
  □ Experts can provide efficient parametrized versions
  □ Non-experts can use them as building blocks for parallel applications

■ Analysis:
  □ Algorithms within a category share characteristics
  □ Scaling behavior, bottlenecks, ...
Example: Communication Patterns

Structured Grid

FFT
## Example: Bottlenecks

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Performance Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Matrix</td>
<td>Computation</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>50% Computation, 50% Memory BW</td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td>Memory Latency</td>
</tr>
<tr>
<td>N-Body</td>
<td>Computation</td>
</tr>
<tr>
<td>Structured Grid</td>
<td>Memory BW</td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td>Memory Latency</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Finite State Machine</td>
<td>Nothing Helps!</td>
</tr>
</tbody>
</table>
Programming Model
Programming Model

Operating System Layer
- Thin virtual machine monitor
- Other features optional

Efficiency Layer
- Low overhead
- For 10% of the best programmers

Productivity Layer
- For the rest of the programming community
- Reuse parallel software from Efficiency Layer
- Efficient Composition (Composition/Coordination Lang.)
Compilers
Two approaches:

- **Auto-parallelizing compilers**
  - Identify parallelizable code segments
  - Is it worth the cost of parallelizing?
  - Select best scheduling policy

- **Auto-tuning of parametrized parallel code**
Two approaches:
- Auto-parallelizing compilers
- Auto-tuning of parametrized parallel code

- Strategy used successfully for sequential code (e.g. ATLAS)
- Select from multiple implementations and parameters
- Performed on each target architecture
Hardware

Communication

Transactional Memory
Hardware: Communication

- Mainly two types of communication:
  - Broadcasts (tree structure)
  - **Local** one-to-one communication

- Idea: reconfigurable interconnect paths
  - MPI leaves most of the heavy lifting to the programmer
  - Study communication patterns at runtime and reconfigure hardware
**Hardware: Transactional Memory**

- **Transaction:** Serializability & Atomicity
- **Goal:** consistent parallel memory modification without locks

![Diagram of transactional memory](image)

- Transaction can be committed successfully if
  - data set not modified by other transactions
  - write set not read by other transactions

- Aborted transactions are rolled back
Hardware: Transactional Memory

shared int counter;

void atomicInc() {
    size_t backoff = BACKOFF_MIN;

    while (true) {
        ST(&counter, LTX(&counter) + 1);
        if (COMMIT())
            break;
        else {
            size_t wait = rand() % (01 << backoff);
            while (wait--);
            if (backoff < BACKOFF_MAX)
                backoff++;
        }
    }
}
Questions?
Thanks for listening!


