The Challenge of Parallelism

Programming parallel processors is one of the challenges of our era

Advanced processors:

- NVIDIA Tegra 2 system on a chip (SoC)
  - Dual-core ARM Cortex A9
  - Integrated GPU. Lots of DSP.
  - 1 GHz.
  - 2 single-precision GFLOPs peak (CPUs only)

- Tilera Tile64
  - 64 processors
  - Each tile has L1, L2, can run OS
  - 443 billion operations/sec.
  - 50-833 MHz
  - 50 Gbytes/sec memory bandwidth

Nvidia Fermi
- 16 cores, 4-way multithreaded,
- 4-wide Superscalar, dual-issue, 3
- 2-wide SIMD (half-pumped)
- 2 MB (16 x 128 KB) Registers, 1
- MB (16 x 64 KB) L1 cache, 0.75 MB L2 Cache

Outline

- What doesn’t work
  - Pieces of the problem … and solution
  - General approach to architecting parallel sw
  - Detail on Structural Patterns
  - Detail on Computational Patterns
  - High-level examples of architecting applications

Assumption #1:
How not to develop parallel code

1. Initial Code
2. Profiler
3. Performance profile
4. Re-code with more threads
5. Fast enough
6. Not fast enough
7. Ship it
8. Lots of failures
   N PE’s slower than 1
Architecting Parallel Software
Keutzer and Mattson

**Steiner Tree Construction Time By Routing Each Net in Parallel**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serial</th>
<th>2 Threads</th>
<th>3 Threads</th>
<th>4 Threads</th>
<th>5 Threads</th>
<th>6 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptec1</td>
<td>1.68</td>
<td>1.68</td>
<td>1.70</td>
<td>1.69</td>
<td>1.69</td>
<td>1.69</td>
</tr>
<tr>
<td>newblue1</td>
<td>1.80</td>
<td>1.80</td>
<td>1.81</td>
<td>1.81</td>
<td>1.81</td>
<td>1.82</td>
</tr>
<tr>
<td>newblue2</td>
<td>2.60</td>
<td>2.60</td>
<td>2.62</td>
<td>2.62</td>
<td>2.62</td>
<td>2.61</td>
</tr>
<tr>
<td>adaptec2</td>
<td>1.87</td>
<td>1.86</td>
<td>1.87</td>
<td>1.88</td>
<td>1.88</td>
<td>1.88</td>
</tr>
<tr>
<td>adaptec3</td>
<td>3.32</td>
<td>3.33</td>
<td>3.34</td>
<td>3.34</td>
<td>3.34</td>
<td>3.34</td>
</tr>
<tr>
<td>adaptec4</td>
<td>3.20</td>
<td>3.20</td>
<td>3.21</td>
<td>3.21</td>
<td>3.21</td>
<td>3.21</td>
</tr>
<tr>
<td>adaptec5</td>
<td>4.91</td>
<td>4.90</td>
<td>4.92</td>
<td>4.92</td>
<td>4.92</td>
<td>4.92</td>
</tr>
<tr>
<td>newblue3</td>
<td>2.54</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
</tr>
<tr>
<td>average</td>
<td>1.00</td>
<td>1.0011</td>
<td>1.0044</td>
<td>1.0049</td>
<td>1.0046</td>
<td>1.0046</td>
</tr>
</tbody>
</table>

**Hint: What is this person thinking of?**

Re-code with more threads

Edward Lee, “The Problem with Threads”

Threads, locks, semaphores, data races

**So What’s the Alternative?**

**Outline**
- What doesn’t work
- Pieces of the problem … and solution
  - General approach to architecting parallel sw
  - Detail on Structural Patterns
  - Detail on Computational Patterns
  - High-level examples of architecting applications
Architecting Parallel Software
Keutzer and Mattson

Principles of SW Design

After 15 years in industry, at one time overseeing the technology of 25 software products, I came to the conclusion that:
- Software architecture >> software environment
- Software environment >> programming language

Software architecture: where we begin

Can be built by one person
Requires
- Minimal modeling
- Simple process
- Simple tools

Grady Booch
Object-Oriented Guru

The progress of Object Oriented Programming

Built most efficiently and timely by a team
Requires
- Modeling
- Well-defined process
- Power tools

Grady Booch
OO Guru

Architectural Styles (Garland and Shaw, 1996)

- Pipe and filter
- Object oriented
- Event based
- Layered
- Agent and repository
- Process control
Civil Architectural Styles

Problem: Build a residential home:
Architectural styles:
• Ranch house
• Victorian
• Colonial
• A-frame
• Bungalow
• Cape cod

Goal – Future sw architecture

Grady Booch
OO Guru

How does Modularity Help?

Modularity helps:
- Architect: Makes overall design sound and comprehensible
- Project manager:
  - As a manager I am able to comfortably assign different modules to different developers
  - I am also able to use module definitions to track development
- Module implementors: As a module implementor I am able to focus on the implementation, optimization, and verification of my module with a minimum of concern about the rest of the design
- Modularity helps us to identify useful invariants and key computations

What’s life like without modularity?

- Spaghetti code
- Wars over the interpretation of the specification
- Waiting on other coders
- Wondering why you didn’t touch anything and now your code broke
- Hard to verify your code in isolation, and therefore hard to optimize
- Hard to parallelize without identifying key computations

- Modularity will help us obviate all these
  Parnas, “On the criteria to be used on composing systems into modules,” CACM, December 1972.
Object-Oriented Programming

Focused on:
- Program modularity
- Data locality
- Architectural styles
- Design patterns

Neglected:
- Application concurrency
- Computational details
- Parallel implementations

Another piece of the puzzle from the HPC community

HPC knows a lot about computations, application concurrency, efficient programming, and parallel implementation

\[
\phi = \frac{\partial}{\partial x} + \frac{1}{2} \frac{\partial}{\partial y} + \frac{\partial}{\partial z} - \nabla \cdot \mathbf{v} = 0
\]

\[
x = \sum_{j=1}^{N} \theta_j + x_j
\]

\[
z = W S_j \{W^* x\}
\]

\[
x = f |P^T y + \mu^T P F x|
\]

minimize \( \|W x\|_1 \) subject to
\( F_0 x = p \)
\( \|G x - z\|_2 < \epsilon \)

\[
J(w) = \int_\Omega \phi \left( f(x + w) - f(x) \right)^2 dx + \gamma \int_\Omega \phi \left( \nabla K(x + w) - \nabla f(x) \right)^2 dx + \alpha \int_\Omega \phi \left( \nabla u^2 + |\nabla v|^2 \right) dx
\]

Pop Quiz: Is a software program more like?

- a) A building
- b) A factory

We need to consider the machinery – but what is the machinery?

Defining Software Requirements for Scientific Computing

Phillip Colella
Applied Numerical Algorithms Group
Lawrence Berkeley National Laboratory
Architecting Parallel Software
Keutzer and Mattson

High-end simulation in the physical sciences consists of seven algorithms:

- Structured Grids (including locally structured grids, e.g. AMR)
- Unstructured Grids
- Fast Fourier Transform
- Dense Linear Algebra
- Sparse Linear Algebra
- Particles
- Monte Carlo

Well-defined targets from algorithmic and software standpoint.

Remainder of this talk will consider one of them (structured grids) in detail.

Par Lab’s contribution: from 7 to 13 families of computations

Well-defined targets from algorithmic and software standpoint.

Remainder of this talk will consider one of them (structured grids) in detail.

Unfortunately ... HPC approach to software architecture

Technically this is known as a monolithic architecture

How can we integrate these insights?

- We wish to find an approach to building software that gives equal support for two key problems of software design – how to structure the software and how to efficiently implement the computations
Architecting Parallel Software
Keutzer and Mattson

Outline

- What doesn’t work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

Alexander’s Pattern Language

Christopher Alexander’s approach to (civil) architecture:
- “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” Page x, A Pattern Language, Christopher Alexander

Alexander’s 253 (civil) architectural patterns range from the creation of cities (2. distribution of towns) to particular building problems (232. roof cap)

A pattern language is an organized way of tackling an architectural problem using patterns

Main limitation:
- It’s about civil not software architecture!!

Uses of Patterns

Patterns give names and definitions to key elements of design

This enables us to better:

- Teach design – a palette of defined design principals
  - Gives new ideas – approaches you may not have considered
  - Gives a set of finiteness – if you’ve considered all the patterns then you can rest assured you’ve considered the key approaches
- Guide design – articulate design decisions succinctly
- Communicate design – improve documentation, facilitate maintenance of software

Patterns capture and preserve bodies of knowledge about key design decisions

- Useful implementation techniques
- Likely challenges/bottlenecks that will come with the use of this pattern (e.g. repository bottleneck in agent and repository)

Architecting Parallel Software with Patterns

Identify the Software Structure
- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Process Control
- Layered Systems
- Model-view controller
- Iterator
- MapReduce
- Arbitrary Task Graphs
- Puppeteer

Identify the Key Compositions
- Graph Algorithms
- Dynamic programming
- Dense/Sparse Linear Algebra
- (Un)Structured Grids
- Graphical Models
- Finite State Machines
- Backtrack Branch-and-Bound
- N-Body Methods
- Circuits
- Spectral Methods
Architecting Parallel Software
Keutzer and Mattson

Architecting Parallel Software

Decompose Tasks
- Group tasks
- Order Tasks

Decompose Data
- Identify data sharing
- Identify data access

Identify the Software Structure
Identify the Key Computations

Identify the SW Structure

Structural Patterns

- Pipe-and-Filter
- Agent-and-Repository
- Event-based coordination
- Iterator
- MapReduce
- Process Control
- Layered Systems

These define the structure of our software but they do not describe what is computed

Analogy: Layout of Factory Plant

Identify key computations ....

Computational patterns describe the key computations but not how they are implemented
Analogy: Machinery of the Factory

Analogy: Architected Factory

Architecting Parallel Software

Structural Patterns
- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Layered Systems
- Model-view-controller
- Arbitrary Task Graphs
- Puppeteer
- Iterator/BSP
- MapReduce

Computational Patterns
- Graph-Algorithms
- Dynamic-Programming
- Dense-Linear-Algebra
- Sparse-Linear-Algebra
- Unstructured-Grids
- Structured-Grids
- Graphical-Models
- Finite-State-Machines
- Backtrack-Branch-and-Bound
- N-Body-Methods
- Circuits
- Spectral-Methods
- Monte-Carlo

Hint: What is this person thinking of?

Re-code with more threads

Edward Lee, “The Problem with Threads”

Threads, locks, semaphores, data races
What’s this person thinking of ...?  
- Need to integrate the insights into computation provided by HPC with the insights into program structure provided by software architectural styles

Outline
- What doesn’t work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

Inventory of Structural Patterns
1. pipe and filter
2. iterator
3. MapReduce
4. blackboard/agent and repository
5. process control
6. Model View Controller
7. layered
8. event-based coordination
9. puppeteer
10. static task graph

Elements of a structural pattern
- Components are where the computation happens
- A configuration is a graph of components (vertices) and connectors (edges)
- A structural patterns may be described as a family of graphs.
- Connectors are where the communication happens
Pattern 1: Pipe and Filter

- Filters embody computation
- Only see inputs and produce outputs
- Pipes embody communication
- May have feedback

Examples?

Examples of pipe and filter

- Almost every large software program has a pipe and filter structure at the highest level

Pattern 2: Iterator Pattern

- Variety of functions performed asynchronously
- Synchronize results of iteration
- Exit condition met?
  - Yes
  - No
  
Examples?
Pattern 3: MapReduce

To us, it means:
- A map stage, where data is mapped onto independent computations
- A reduce stage, where the results of the map stage are summarized (i.e. reduced)

Examples?

Examples of Map Reduce

- General structure:
  - Map a computation across distributed data sets
  - Reduce the results to find the best/(worst), maxima/(minima)

- Support-vector machines (ML)
  - Map to evaluate distance from the frontier
  - Reduce to find the greatest outlier from the frontier

- Speech recognition
  - Map HMM computation to evaluate word match
  - Reduce to find the most-likely word sequences

Pattern 4: Agent and Repository

Agent and repository: Blackboard structural pattern

Agents cooperate on a shared medium to produce a result

Key elements:
- Blackboard: repository of the resulting creation that is shared by all agents (circuit database)
- Agents: intelligent agents that will act on blackboard (optimizations)
- Manager: orchestrates agents access to the blackboard and creation of the aggregate results (scheduler)

Examples?

Example: Compiler Optimization

Optimization of a software program

- Intermediate representation of program is stored in the repository
- Individual agents have heuristics to optimize the program
- Manager orchestrates the access of the optimization agents to the program in the repository
- Resulting program is left in the repository
Example: Logic Optimization

- Optimization of integrated circuits
- Integrated circuit is stored in the repository
- Individual agents have heuristics to optimize the circuitry of an integrated circuit
- Manager orchestrates the access of the optimization agents to the circuit repository
- Resulting optimized circuit is left in the repository

Pattern 5: Process Control

- Process control:
  - Process: underlying phenomena to be controlled/computed
  - Actuator: task(s) affecting the process
  - Sensor: task(s) which analyze the state of the process
  - Controller: task which determines what actuators should be effected

Examples of Process Control

Examples of Process Control:
Matrix Multiply Autotuner (Michael Driscoll)

- Objective Function (e.g. minimize energy)
- Problem Dimensions (M,N,K)
- Source Code
- Watts Up Pro Power Meter
Pattern 9: Puppeteer

- Need an efficient way to manage and control the interaction of multiple simulators/computational agents
- Puppeteer Pattern – guides the interaction between the tasks/puppets to guarantee correctness of the overall task
- Puppeteer: 1) schedules puppets 2) manages exchange of data between puppets
- Difference with agent and repository?
  - No central repository
  - Data transfer between tasks/puppets

Video Game

Model of circulation

- Modeling of blood moving in blood vessels
- The computation is structured as a controlled interaction between solid (blood vessel) and fluid (blood) simulation codes
- The two simulations use different data structures and the number of iterations for each simulation code varies
- Need an efficient way to manage and control the interaction of the two codes

Outline

- What doesn’t work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications
You explore these every class

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack / B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monte Carlo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outline

- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

Large Vocabulary Continuous Speech Recognition

- Inference engine based system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan) [10,15,9]
- Modular and flexible setup
  - Shown to be effective for Arabic, English, Japanese, and Mandarin

LVCSR Software Architecture
Key computation: HMM Inference Algorithm

- Finds the most-likely sequence of states that produced the observation

\[ m[t][s_t] = \max_{k \in \text{Frontier}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t) \]

Viterbi Algorithm

Legends:
- A State
- An Observation
- \( P(s_t|s_{t-1}) \)
- \( P(x_t|s_t) \)

Markov Condition:
\[ m[t][s_t] = \max_{k \in \text{Frontier}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t) \]

Inference Engine in LVCSR

- Three steps of inference
  0. Gather operands from irregular data structure to runtime buffer
  1. Perform observation probability computation
  2. Perform graph traversal computation

Parallelism in the inference engine:

1. \( P(x_t|s_t) \)
2. \( m[t][s_t] \)

Each Filter is a Map Reduce

0. Gather operands

\[ m[t][s_t] = \max_{k \in \text{Frontier}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t) \]

- Gather and coalesce each of the above operands for every \( s_t \)
- Facilitates opportunity for SIMD

Each Filter is Map Reduce

1. Observation probability computation

\[ m[t][s_t] = \max_{k \in \text{Frontier}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t) \]

- Gaussian Mixture Model Probability
- Probability that given this feature-frame (e.g. 10ms) we are in this state/phone

max
Each Filter is Map Reduce
2. graph traversal computation

- **Map** probability computation across distributed data sets – perform multiplication as below
- **Reduce** the results to find the maximum likely states

\[ m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(y_t|s_t) \]

HMM computed with Dynamic Programming

```
Observations | Tag | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17
-------------|----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
Wreck        | a  | n | e | c | i | s | b | e | a | c | h | h | c | h | c | h | c | h
```

This Approach Works

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRI</td>
<td>100x</td>
</tr>
<tr>
<td>SVM-train</td>
<td>&gt;2900 Downloads</td>
</tr>
<tr>
<td>SVM-classify</td>
<td>&gt;2900 Downloads</td>
</tr>
<tr>
<td>Contour</td>
<td>&gt;2900 Downloads</td>
</tr>
<tr>
<td>Object Recognition</td>
<td>80x</td>
</tr>
<tr>
<td>Poselet</td>
<td>20x</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>32x</td>
</tr>
<tr>
<td>Speech</td>
<td>11x</td>
</tr>
<tr>
<td>Value-at-risk</td>
<td>60x</td>
</tr>
<tr>
<td>Option Pricing</td>
<td>25x</td>
</tr>
</tbody>
</table>

Outline

- What doesn't work
- Pieces of the problem ... and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications
- Summary

Outline

- What doesn't work
- Pieces of the problem ... and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications
- Summary

Recap: Architecting Parallel Software

1. Start with a compelling, performance sensitive application.

2. Define the overall structure

3. Define computations inside structural elements

4. Compose Structural and computational patterns to yield software architecture

"Image Features Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC'09


Computational Patterns

- Make me Feel Smart

- For many years computation has been like a big ball of yarn
- Computational patterns help us to unravel it into 13 strands
- Alan Kay “Perspective is worth 100 IQ points.”
- Computational patterns give us perspective on computation
Structural Patterns
Make me Feel Organized

- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Layered Systems
- Model-view-controller
- Arbitrary Task Graphs
- Puppeter
- Iterator/BSP
- MapReduce

• The modularity provided by structural patterns make me feel organized.
• Even the most complex application can be broken down into manageable modules

Summary

- The key to productive and efficient parallel programming is creating a good software architecture – a hierarchical composition of:
  - Structural patterns: enforce modularity and expose invariants
    - I showed you five – five more will be all you need
  - Computational patterns: identify key computations to be parallelized
  - Orchestration of computational and structural patterns creates architectures which greatly facilitates the development of parallel programs:

Patterns: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns
PALLAS: http://parlab.eecs.berkeley.edu/research/pallas

More examples
Architecting Speech Recognition

Feature Extraction

Image histograms are common to many feature extraction procedures, and are an important feature in their own right:

- Agent and Repository: Each agent computes a local transform of the image, plus a local histogram.
- Results are combined in the repository, which contains the global histogram.
- The data dependent access patterns found when constructing histograms make them a natural fit for the agent and repository pattern.

CBIR Application Framework

Train Classifier: SVM Training

Exercise Classifier: SVM Classification

- SVM
- Compute dot products
- Dense Linear Algebra
- Compute Kernel values, summ & scale
- MapReduce

Assumption #2: This won’t help either

Code in new cool language

Profiler

Performance profile

Re-code with cool language

Not fast enough

Fast enough

After 200 parallel languages where’s the light at the end of the tunnel?

Key Elements of Kurt’s SW Education

- AT&T Bell Laboratories: CAD researcher and programmer
  - Algorithms: D. Johnson, R. Tarjan
  - Programming Pearls: S C Johnson, K. Thompson, (Jon Bentley)
- Developed useful software tools:
  - Plaid: programmable logic aid: used for developing 100’s of FPGA-based HW systems
  - CONES/DAAGON: used for designing >30 application-specific integrated circuits
- Synopsys: researcher
- Software architecture: Randy Allen, Albert Wang
- High-level Invariants: Randy Allen, Albert Wang
- Berkeley: teaching software engineering and Par Lab
  - Took the time to reflect on what I had learned:
- Architectural styles: Garlan and Shaw

Parallel Programming environments in the 90’s

- Cilk
- Charm++
- Charm
- SISCAL
- COOL
- Cilk
- SuperPascal
- C++
- Smalltalk
Assumption #3: Nor this

Initial Code

Super-compiler

Tune compiler

Performance profile

Not fast enough

Fast enough

Ship it

30 years of HPC research don't offer much hope

Automatic parallelization?

Aggressive techniques such as speculative multithreading help, but they are not enough.

Ave SPECint speedup of 8%... will climb to ave. of 15% once their system is fully enabled.

There are no indications auto par. will radically improve any time soon.

Hence, I do not believe Auto-par will solve our problems.

Results for a simulated dual core platform configured as a main core and a core for speculative execution.

Reinvention of design?

- In 1418 the Santa Maria del Fiore stood without a dome.
- Brunelleschi won the competition to finish the dome.
- Construction of the dome without the support of flying buttresses seemed unthinkable.

http://www.templejc.edu/dept/Art/ASmith/ARTS1304/Joel/ZoomSlide0010.html

Innovation in architecture

- After studying earlier Roman and Greek architecture, Brunelleschi drew on diverse architectural styles to arrive at a dome design that could stand independently.

http://www.templejc.edu/dept/Art/ASmith/ARTS1304/Joel/ZoomSlide0010.html
Innovation in tools

- His construction of the dome design required the development of new tools for construction, as well as an early (the first?) use of architectural drawings (now lost).

Scaffolding for cupola
Mechanism for raising materials
http://www.artist-biography.info/gallery/filippo_brunelleschi/67/

Innovation in use of building materials

- His construction of the dome design also required innovative use of building materials.

Herringbone pattern bricks
http://www.buildingstonemagazine.com/winter-06/art/dome8.jpg

Resulting Dome

Completed dome
http://www.duomofirenze.it/storia/cupola_eng.htm

The point?

- Challenges to design and build the dome of Santa Maria del Fiore showed underlying weaknesses of architectural understanding, tools, and use of materials.
- By analogy, parallelizing code should not have thrown us for such a loop. Our difficulties in facing the challenge of developing parallel software are a symptom of underlying weakness in our abilities to:
  - Architect software
  - Develop robust tools and frameworks
  - Re-use implementation approaches
- Time for a serious rethink of all of software design