Patterns in parallel computing

Oregon State University EECS
June 4, 2014

Michael Wrinn, Ph.D.
Research Director
University Research Office, Intel

Disclaimer: the views expressed in this talk are those of the speaker, not his employer. This is an academic style talk, not intended to address details of any particular Intel product or plan.

Example Problem: Numerical Integration

Mathematically, we know that:

\[ \int_0^1 \frac{4.0}{(1+x^2)} \, dx = \pi \]

We can approximate the integral as a sum of rectangles:

\[ \sum_{i=0}^{n} F(x_i) \Delta x \approx \pi \]

Where each rectangle has width \( \Delta x \) and height \( F(x_i) \) at the middle of interval \( i \).

Parallel Pi: OpenMP solution

```c
#include <omp.h>

static long num_steps = 100000;
double step = 1.0/(double) num_steps;

void main ()
{
    int i;
    double x, pi, sum = 0.0;

    for (i=0;i<num_steps; i++)
    {
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }

    pi = step * sum;
}
```

Akari Algorithm

Input: board size and list of number and black squares
Sort numbered squares by value; plain black squares after numbered
Place rocks around all "4" squares
• Get next numbered square in list
• Try all rock combinations around square, via recursive call
  - "3" square => 4 combinations
  - "2" square => 6 combinations
  - "1" square => 4 combinations
• If no more numbered squares, compile list of all open squares
  Using backtracking:
    - Try rock in/out next open square from list
    - Solution reached when no more open squares

Case Study: Akari
Logic puzzle from Nikoli
Goal: Place chess rooks on open squares such that
• No two rooks attack each other
• Numbered squares surrounded by specified number of rooks
• All open squares are "covered" by one or more rooks
• Black squares block attack of rooks

Input:

```
Initial Configuration
3 1 0
2 4 0

Solution Configuration
3 1 1
2 4 0
```

Output:
Search Tree – placeThree() example

Conventional analysis: where to parallelize?

What might Hotspot Analysis show?
- If move generation is fast, very little time in BT()
- Likely DeadEnd() or Solution() (at leaf) of search tree consume most execution

Example: Akari solver
- countBlanks() is 96% of serial time
- Simple for-loop over board squares
  - 360 squares for dataset used

Architecting Parallel Software: Design patterns in practice

A brief look at work developed by
- Kurt Keutzer, UC Berkeley
- Tim Mattson, Intel
- many contributors at UC Berkeley, UIUC, Intel

How do we fix the parallel programming problem?

- Focus on software architecture and time-tested designs that have been shown to work...
  - Given a good design, a programmer can create quality software regardless of the language
  - Design patterns are a way to put these approaches into writing.

Our Pattern Language 2011

Structural Patterns
- Pipe and Filter
- Agent and Repository
- Process Control
- Event-Driven, Implicit
- P-process

Algorithm Strategy Patterns
- Task Parallelism
- Recursive splitting

Implementation Strategy Patterns
- Data Parallelism
- Geometric Decomposition
- Graph partitioning
- Task Graph
- Shared Data
- Data structure

Parallel Execution Patterns
- Message Passing
- Distributed Array
- Collective
- Task-Graph
- Dual-MON temporal

Computational Patterns
- Computational Geometric Decomposition
- Geometrical Models
- Circuits
- Special Methods
- Monte Carlo
Our Pattern Language

Applications
- Structural Patterns
- Computational Patterns
- Parallel Algorithm Strategy Patterns
- Implementation Patterns
- Execution Patterns

Identify the SW Structure

Structural Patterns
- Pipe-and-Filter
- Agent-and-Repository
- Process-Control
- Event-Based/Implicit-Invocation
- Puppeteer
- Model-View-Controller
- Iterative-Refinement
- Map-Reduce
- Layered-Systems
- Arbitrary-Static-Task-Graph

These define the software structure but do not describe what is computed.

Analogy: Layout of Factory Plant

Analogy: Machinery of the Factory

Structural Pattern: Pipe and Filter

- Filters embody computation
- Only see inputs and produce outputs
- No global or shared state

Examples: pipe and filter

- Almost every large software program has a pipe and filter structure at the highest level
- Filter 1
- Filter 2
- Filter 3
- Filter 4
- Filter 5
- Filter 6
- Filter 7

Examples:
Structural Pattern: 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 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

Identify Key Computations

- Apps
  - Finite State Mach.
  - Dwarves
  - SPEC
  - Games
  - ML
  - HPC
  - CAD
  - Health
  - Image
  - Speech
  - Music

These define the key computations, but do not describe how they are implemented

Parallel Algorithm Strategy Patterns

- Parallel Algorithm strategies:
  - These patterns define high-level strategies to exploit concurrency within a computation for execution on a parallel computer.
  - They address the different ways concurrency is naturally expressed within a problem/application.

How does the software architecture map onto parallel algorithms?
Implementation Strategy Patterns

- Implementation strategies:
  - These are the structures that are realized in source code to support (a) how the program itself is organized and (b) common data structures specific to parallel programming.

How do parallel algorithms map onto source code in a parallel programming language?

Parallel Execution Patterns

- Parallel Execution Patterns:
  - These are the approaches often embodied in a runtime system that supports the execution of a parallel program.

How is the source code realized as an executing program running on the target parallel processor?

Compelling Application: Fast, Robust Pediatric MRI

- Pediatric MRI is difficult:
  - Children cannot sit still, breathhold
  - Low tolerance for long exams
  - Anesthesia is costly and risky
  - Like to accelerate MRI acquisition

- Advanced MRI techniques exist, but require data- and compute-intensive algorithms for image reconstruction

- Reconstruction must be fast, or time saved in accelerated acquisition is lost in computing reconstruction

- Slow reconstruction times are a non-starter for clinical use

SW architecture of image reconstruction

- 100X faster reconstruction
- Higher-quality, faster MRI
- This image: 8 month-old patient with cancerous mass in liver
  - 256 x 154 x 8 data size
  - Serial Recon: 1 hour
  - Parallel Recon: 1 minute
  - Fast enough for clinical use
- Software currently deployed at Lucile Packard Children’s Hospital for clinical study of the reconstruction technique
### Software Design Patterns: earlier lesson

**Early days OO**
- Perception: Object-oriented? Isn’t that just an academic thing?
- Usage: specialists only. Mainstream regards with indifference or anxiety.
- Performance: not so good.

**Now**
- Perception: OO=programming. Isn’t that just an HPC thing?
- Usage: cosmetically widespread, some key concepts actually deployed.
- Performance: so-so, masked by CPU advances until now.

### Parallel Design Patterns: does history rhyme? (a prediction from 2011)

**Then**
- Perception: Parallel programming? Isn’t that just an HPC thing?
- Usage: specialists only. Mainstream regards with indifference or anxiety.
- Performance: very good, for the specialists.

**Soon**
- Perception: PP=programming. Isn’t this how it was always done?
- Usage: widespread, key concepts actually deployed.
- Performance: broadly sufficient. Application domains greatly expanded.

### Computational Characteristics of (big data) application areas (2014)

**Patterns**
- Graph Algorithms
- Graphical Models
- Backtracking / BB
- Finite State Machines
- Metaheuristics
- Dynamic Programming
- N-Body
- Unstructured Grid
- D-Hull (FPC)
- Monte Carlo

**Apps**
- Data Curating
- Online
- Big Data Analytics
- Visualization
- User Interface
- Heterogeneous
- Acceleration
- Memory
- Software

### DNA Pipeline: BWA+GATK: Whole Genome Sample: ~65x Coverage

**Overview**
- Big Data starts here
- Systems Biology Development of personalized treatments
- Genomics - Architecture for doing all in a day

**SW - HW**
- 30-36 hours

**Improvement**
- 6X improvement so far and 4X without major code change and rest with code changes.

### Genomics - Architecture for doing all in a day

April 31, 2014

Gans Srinivasu, Sr. Principal Engineer

Karthik Srinivas, Sr. Principal Engineer

Paolo Ghodrat, Sr. Principal Engineer

Mishali Narvaez, Sr. Principal Engineer

Engineer - Team HSS

- Redesign of algorithms and domain models
- Integration of software packages and tools
- Performance and scalability improvements
- Improved data quality and analysis

**DNA Pipeline**
- BWA + GATK
- Whole Genome Sample
- ~65x Coverage

**Key Concepts**
- Parallelism
- Thread-level parallelism
- Process-level parallelism
- Memory constraints
- Software and algorithms

**Heterogeneous Computing**
- Accelerators + GPU + Memory computing / data store

**SW**
- 30-36 hours
Pair HMM Acceleration using AVX

- Computation kernel and bottleneck in GATK Haplotype Caller
- AVX enables 8 floating point SIMD operations in parallel
- 2 Ways to vectorize HMM computation
  - Intra-Sequence – Parallelize computation within one HMM matrix operation. Run multiple (8) computations concurrently along diagonal
  - Inter-Sequence – Perform multiple (8) HMM matrix operations at once

<table>
<thead>
<tr>
<th></th>
<th>Time (seconds)</th>
<th>Speedup C++/Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial C++</td>
<td>1540</td>
<td>1x / 9x</td>
</tr>
<tr>
<td>1 core with AVX (Intra)</td>
<td>340</td>
<td>4.5x / 40.7x</td>
</tr>
<tr>
<td>1 core with AVX (Inter)</td>
<td>285</td>
<td>5.4x / 48.6x</td>
</tr>
<tr>
<td>24 cores with AVX (Inter)</td>
<td>14.3</td>
<td>108x / 970x</td>
</tr>
<tr>
<td>24 cores hybrid (Inter)</td>
<td>15.7</td>
<td>98x / 882x</td>
</tr>
</tbody>
</table>
Graph abstraction: essential for data driven problems

Flashback to 1998

First Google advantage: a Graph Algorithm & a System to Support it!

Example 1: PageRank

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>5.5 hrs</td>
</tr>
<tr>
<td>Twister</td>
<td>1 hr</td>
</tr>
<tr>
<td>GraphLab</td>
<td>8 min</td>
</tr>
</tbody>
</table>

40M Webpages, 1.4 Billion Links

Hadoop results from Kang et al. '11
Twister (in-memory MapReduce) [Ekanayake et al. '10]

Example 2: Never Ending Learner Project (CoEM)

<table>
<thead>
<tr>
<th>Method</th>
<th>Cores</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>Distributed GraphLab</td>
<td>32 EC2 machines</td>
<td>80 secs</td>
</tr>
</tbody>
</table>

2 orders of mag faster ➔ 2 orders of mag cheaper

Example 3: Triangle Counting on Twitter Graph

<table>
<thead>
<tr>
<th>Method</th>
<th>Machines</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>1636</td>
<td>423 Min</td>
</tr>
<tr>
<td>GL2</td>
<td>64</td>
<td>15 Secs</td>
</tr>
<tr>
<td>PowerGraph</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

34.8 Billion Triangles

Why?? Wrong Abstraction!
(Broadcast O(degree^2) messages per Vertex)

Suri and S. Vassilvitski, “Counting triangles and the curse of the last reducer,” WWW’11

Wrapping up (pontificating)...

Patterns:
• use them
• use the right ones
• beware hasty/naive parallelization (pattern understanding will help you here)

Patterns facilitate good design; their utility continues to grow.