Using Design Patterns to help make everyone a parallel programmer

Tim Mattson, Principle Engineer, Intel labs
timothy.g.mattson@intel.com
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY
THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY
EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY
OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY,
OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY
RIGHT.

Software and workloads used in performance tests may have been optimized for performance
only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are
measured using specific computer systems, components, software, operations and
functions. Any change to any of those factors may cause the results to vary. You should
consult other information and performance tests to assist you in fully evaluating your
contemplated purchases, including the performance of that product when combined with
other products.

Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for
optimizations that are not unique to Intel microprocessors. These optimizations include SSE2,
SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the
availability, functionality, or effectiveness of any optimization on microprocessors not
manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for
use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are
reserved for Intel microprocessors. Please refer to the applicable product User and Reference
Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Disclaimer

• The views expressed in this talk are mine ... not Intel’s.
• I didn’t do all the work described in this talk. If I say something stupid, blame me ... not the smart people I work with.
• You may be unable to reproduce any performance numbers quoted in this talk
• I will say little to nothing about Intel products ... I am not announcing ANY new products or research programs.

Slides marked with this symbol were produced-with or borrowed-from colleagues at the UC Berkley Par Lab
Outline

• Many-core processors … the new HW/SW contract
• Responding to the new HW/SW contract …
  – Parallel Programming Languages?
  – “Good enough” Standards plus tools that work.
• Making all programmers parallel programmers:
  – A discipline of design: design patterns and frameworks
  – High performance through SW transformation tools
In 1965, Intel co-founder Gordon Moore predicted (from just 3 data points!) that semiconductor density would double every 18 months.

- He was right! Transistors are still shrinking as he projected.
The good old days …

(SPECint) Uniprocessor Performance


Third party names are the property of their owners.
The Hardware/Software contract

• Write your software as you choose and the HW-geniuses will take care of performance.

The result: Generations of performance ignorant software engineers using performance-handicapped languages (such as Java) … which was OK since performance was a HW job.

Third party names are the property of their owners.
... But that world ran into the performance wall

\[ \text{power} = \text{perf}^{1.74} \]

Growth in power is unsustainable

Source: E. Grochowski of Intel
... partial solution: simple low power cores

Mobile CPUs with shallow pipelines use less power

... The rest of the solution: “Moore’s law doubling” of cores


Source: Vishwani Agrawal
Timeline of Many-Core at Intel

**2004**
- Many-core technology Strategic Planning

**2005**
- Many-core R&D agenda & BU Larrabee development

**2006**
- Tera-scale computing research program (80+ projects)

**2007**
- Workloads, simulators, software & insights from Intel Labs

**2008**
- Universal Parallel Computing Research Centers

**2009**
- 1 Teraflops SGEMM on Larrabee @ SC’09¹

**2010**
- Many-core applications research community

**2011**
- Aubrey Isle & Intel® MIC Architecture

**2012**
- Intel® Xeon Phi™ Coprocessor enters Top500 at #150 (pre-launch)²

¹ Source: Intel Measured/demonstrated at SC ’09, Nov. 2009. ² Source: [www.top500.org](http://www.top500.org) June 2012

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance)
The result...

A new contract ... HW people will do what’s natural for them (lots of simple cores) and SW people will have to adapt (rewrite everything)

The problem is this was presented as an ultimatum ... nobody asked us if we were OK with this new contract ... which is kind of rude.
The many core challenge

- A harsh assessment …
  - We have turned to multi-core chips not because of the success of our parallel software but because of our failure to continually increase CPU frequency.

  - Result: a fundamental and dangerous (for the computer industry) mismatch
    - Parallel hardware is ubiquitous.
    - Parallel software is rare

- The Many Core challenge …
  - Parallel software must become as common as parallel hardware

Fortunately, we don’t have to start over “from scratch”. We can draw from past experience with parallelism from high performance computing
Outline

• Many-core processors … the new HW/SW contract

• Responding to the new HW/SW contract …
  – Parallel Programming Languages?
  – “Good enough” Standards plus tools that work.

• Making all programmers parallel programmers:
  – A discipline of design: design patterns and frameworks
  – High performance through SW transformation tools
We tried to solve the programmability problem by searching for the right programming environment in the 90's. Parallel programming environments in the 90's

| ABCPL | ACE | ACT++ | Active messages | Adl | Adsmith | ADDAP | AFAPI | ALWAN | AM | AMDC | AppLeS | Amoeba | ARTS | Athapascan-0b | Aurora | Automap | bb_threads | Blaze | BSP | BlockComm | C* | "C* in C | C** | CarlOS | Cashmere | C4 | CC++ | Chu | Charlotte | Charm | Charm++ | Cid | Cilk | CM-Fortran | Converse | Code | COOL |
|-------|-----|-------|------------------|-----|---------|-------|-------|-------|----|-----|--------|--------|------|----------------|--------|---------|-------------|-------|-----|-----------|----|------|-------|---------|-------|--------|----|------|--------|---------|-------|-----|-----|-----|-----|
|       |     |       |                  |     |         |       |       |       |   |    |        |        |      |                |        |         |              |     |    |           |    |      |       |        |       |        |    |     |        |         |       |    |    |    |    |
|       |     |       |                  |     |         |       |       |       |   |    |        |        |      |                |        |         |              |     |    |           |    |      |       |        |       |        |    |     |        |         |       |    |    |    |    |
A warning I’ve been making for the last 10 years

Is it bad to have so many languages? Too many options can hurt you:
- The Draeger Grocery Store experiment consumer choice:
  - Two Jam-displays with coupon’s for purchase discount.
    - 24 different Jam’s
    - 6 different Jam’s
  - How many stopped by to try samples at the display?
  - Of those who “tried”, how many bought jam?

Programmers don’t need a glut of options ... just give us something that works OK on every platform we care about. Give us a decent standard and we’ll do the rest.

The findings from this study show that an extensive array of options can at first seem highly appealing to consumers, yet can reduce their subsequent motivation to purchase the product.

My optimistic view from 2005 ...

We’ve learned our lesson ... for application developers, they only needed to consider this small set of parallel programming systems.

Parallel Programming API’s today

- Thread Libraries
  - Win32 API
  - POSIX threads.
- Compiler Directives
  - OpenMP - portable shared memory parallelism.
- Message Passing Libraries
  - MPI - message passing
- Coming soon ... a parallel language for managed runtimes? Java or X10?

We don’t want to scare away the programmers ... Only add a new API/language if we can’t get the job done by fixing an existing approach.

Third party names are the property of their owners.
But we didn’t learn our lesson
History is repeating itself!

A small sampling of Programming environments from the NEW golden age of parallel programming (from the literature 2010-2012)

<table>
<thead>
<tr>
<th>AM++</th>
<th>Copperhead</th>
<th>ISPC</th>
<th>OpenACC</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArBB</td>
<td>CUDA</td>
<td>Java</td>
<td>PAMI</td>
<td>SIAL</td>
</tr>
<tr>
<td>BSP</td>
<td>DryadOpt</td>
<td>Liszt</td>
<td>Parallel Haskell</td>
<td>STAPL</td>
</tr>
<tr>
<td>C++11</td>
<td>Erlang</td>
<td>MapReduce</td>
<td>ParalleX</td>
<td>STM</td>
</tr>
<tr>
<td>C++AMP</td>
<td>Fortress</td>
<td>MATE-CG</td>
<td>PATUS</td>
<td>SWARM</td>
</tr>
<tr>
<td>Charm++</td>
<td>GA</td>
<td>MCAPI</td>
<td>PLINQ</td>
<td>TBB</td>
</tr>
<tr>
<td>Chapel</td>
<td>GO</td>
<td>MPI</td>
<td>PPL</td>
<td>UPC</td>
</tr>
<tr>
<td>Cilk++</td>
<td>Gossamer</td>
<td>NESL</td>
<td>Pthreads</td>
<td>Win32 threads</td>
</tr>
<tr>
<td>CnC</td>
<td>GPars</td>
<td>OoOJava</td>
<td>PXIF</td>
<td>X10</td>
</tr>
<tr>
<td>coArray Fortran</td>
<td>GRAMPS</td>
<td>OpenMP</td>
<td>PyPar</td>
<td>XMT</td>
</tr>
<tr>
<td>Codelets</td>
<td>Hadoop</td>
<td>OpenCL</td>
<td>Plan42</td>
<td>ZPL</td>
</tr>
<tr>
<td></td>
<td>HMMP</td>
<td>OpenSHMEM</td>
<td>RCCE</td>
<td></td>
</tr>
</tbody>
</table>

Note: I’m not criticizing these technologies. I’m criticizing our collective urge to create so many of them.

Third party names are the property of their owners.
But we didn’t learn our lesson
History is repeating itself!

A small sampling of Programming environments from the NEW golden age of parallel programming (from the literature 2010-2012)

<table>
<thead>
<tr>
<th>AM++</th>
<th>Copperhead</th>
<th>ISPC</th>
<th>OpenACC</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArBB</td>
<td>CUDA</td>
<td>Java</td>
<td>PAMI</td>
<td>SIAL</td>
</tr>
<tr>
<td>BSP</td>
<td>DryadOpt</td>
<td>Liszt</td>
<td>Parallel Haskell</td>
<td>STAPL</td>
</tr>
<tr>
<td>C++11</td>
<td>Erlang</td>
<td>MapReduce</td>
<td>ParalleX</td>
<td>STM</td>
</tr>
<tr>
<td>C++AMP</td>
<td>Fortress</td>
<td>MATE-CG</td>
<td>PATUS</td>
<td>SWARM</td>
</tr>
<tr>
<td>Charm++</td>
<td>GA</td>
<td>MCAPI</td>
<td>PLINQ</td>
<td>TBB</td>
</tr>
<tr>
<td>Chapel</td>
<td>GO</td>
<td>MPI</td>
<td>PPL</td>
<td>UPC</td>
</tr>
<tr>
<td>Cilk++</td>
<td>Gossamer</td>
<td>NESL</td>
<td>Pthreads</td>
<td>Win32 threads</td>
</tr>
<tr>
<td>CnC</td>
<td>GParS</td>
<td>OoOJava</td>
<td>PXIF</td>
<td>X10</td>
</tr>
<tr>
<td>coArray Fortran</td>
<td>GRAMPS</td>
<td>OpenMP</td>
<td>PyPar</td>
<td>XMT</td>
</tr>
<tr>
<td>Codelets</td>
<td>Hadoop</td>
<td>OpenCL</td>
<td>Plan42</td>
<td>ZPL</td>
</tr>
<tr>
<td></td>
<td>HMMP</td>
<td>OpenSHMEM</td>
<td>RCCE</td>
<td></td>
</tr>
</tbody>
</table>

Note: I’m not criticizing these technologies. I’m criticizing our collective urge to create so many of them.

Prog. Envs. In **RED** are being pushed to application programmers by vendors.

Third party names are the property of their owners.
Two paths to designing SW technology:

- MIT model … (e.g. Scheme and Common Lisp)
  1. Simple, elegant, productive interface.
  2. Implementation should be simple, but **NOT** if it complicates item 1
  3. Correct and completeness are essential … even if they complicate point 2.

- New Jersey model … (e.g. AT&T labs with Unix and C)
  1. Simple, elegant, productive interface
  2. Implementation **MUST** be simple, **EVEN IF** it complicates item 1
  3. Correct and complete … but **NOT** if it complicates point 2.

The MIT model is “the right thing”. The new Jersey model is “worse”.

Gabriel concludes in his now famous essay that

**Worse is better**

… with “the worse” you get a solution “out there” for people to use instead of endlessly waiting for “the right thing”. And over time “the worse” becomes the “familiar” and evolves to “get the job done”.
High Performance Parallel Programming –
One Toolset from Multicore to Many-core to Heterogeneous Computing

Scale Forward Tomorrow.
Intel® Software Development Products –
High Performance Tools

Industry-leading performance from advanced compilers

Comprehensive Libraries

Parallel Programming Models

Insightful Performance Analysis Tools

Intel® C/C++ and Fortran Compilers
 w/OpenMP and OpenCL

Intel® MKL, Intel® IPP Library

Intel® Cilk Plus, Intel® TBB Library

Intel® VTune™ Amplifier XE
 Intel® Inspector XE
 Intel® Advisor

Intel® MPI Library

Intel® Trace Analyzer and Collector

Performance. Scale Forward.
Intel® Software Development Products
Performance – Scalability – Efficiency

Meet us at OOP 2014 Exhibition Floor

System Developers
C/C++ developers targeting Intel® Processors based Embedded or Mobile devices

High Performance
C/C++ and Fortran developers Windows* and Linux*
High performance, cross platform apps

High Performance on MPI Cluster
C/C++ and Fortran developers Windows* and Linux*
High performance MPI clusters

HTML5
HTML5 Cross-Platform App Development
Outline

• Many-core processors … the new HW/SW contract
• Responding to the new HW/SW contract …
  – Parallel Programming Languages?
  – “Good enough” Standards plus tools that work.
• Making all programmers parallel programmers:
  – A discipline of design: design patterns and frameworks
  – High performance through SW transformation tools
Concurrency in Parallel software:

Find and Expose Concert tasks

Organize tasks for Parallel execution

Express concurrency in the Source code

Original Problem

Tasks, shared and local data

Units of execution + new shared data for extracted dependencies

Corresponding source code

Program SPMD_Emb_Par()
{
    TYPE *tmp, *func();
    global_array Data(TYPE);
    global_array Res(TYPE);
    int Num = get_num_procs();
    int id = get_proc_id();
    if (id==0) setup_problem(N, Data);
    for (int I= 0; I<N;I=I+Num){
        tmp = func(I, Data);
        Res.accumulate(tmp);
    }
}
Maybe all we need to do is teach programmers how to write parallel code with existing languages.

Most Parallel Programs use a modest number of algorithm structures.
Maybe all we need to do is teach programmers how to write parallel code with existing languages.
Fork-Join Pattern

- Master thread spawns a team of threads as needed.
- Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program.
Data Parallelism Pattern

Use when:
- Your problem is defined in terms of collections of data elements operated on by a similar (if not identical) sequence of instructions; i.e. the concurrency is in the data.

Solution
- Define collections of data elements that can be updated in parallel.
- Define computation as a sequence of collective operations applied together to each data element.

Often supported with the data-parallel/Index-map pattern.

OpenCL, CUDA

Third party names are the property of their owners.
The SPMD Design Pattern

A sequential program working on a data set

- Replicate the program.
- Add glue code
- Break up the data

• A single program working on a decomposed data set.

• Use Node ID and numb of nodes to split up work between processes.

• Coordination by passing messages.

Third party names are the property of their owners.
Bulk Synchronous Processing

Many MPI applications have few (if any) sends and receives. They use a design pattern called "Bulk Synchronous Processing".

- Uses the Single Program Multiple Data pattern
- Each process maintains a local view of the global data
- A problem broken down into phases each composed of two subphases:
  - Compute on local view of data
  - Communicate to update global view on all processes (collective communication).
- Continue phases until complete

Third party names are the property of their owners.
Divide and conquer

Split the problem into smaller sub-problems. Continue until the sub-problems can be solved directly.

3 Options:
- Do work as you split into sub-problems.
- Do work only at the leaves.
- Do work as you recombine.

Third party names are the property of their owners.
Computer Games: one of the few (only?) consumer SW industries that have successfully embraced many-core industry-wide

- The key: Enforce a separation of concerns:
  - A small number (<10%) of high priced “Technology programmers” optimize the game engine for specific platforms (C, assembly, etc)
  - The rest of the team focuses on artwork, the story line and putting the components together to create the final product. (C++, scripting languages and frameworks)

This is a trend reaching across the software industry ...
  - Programming by importing functionality from existing modules
  - Scripting languages specialized through custom modules to increase productivity across a software team.
  - Frameworks ... Partial solutions that are specialized to solve specific problems in a domain.

Third party names are the property of their owners.
Par Lab Research Overview

**Easy to write correct software that runs efficiently on manycore**

- **Applications**
  - Personal Health
  - Image Retrieval
  - Hearing, Music
  - Speech
  - Parallel Browser

- **Design Pattern Language (OPL)**
- **Composition & Coordination Language (C&CL)**
- **C&CL Compiler/Interpreter**
  - Parallel Libraries
  - Parallel Frameworks

- **Efficiency Languages**
- **Sketching**
- **Autotuners**
- **Legacy Code**
- **Schedulers**
- **Communication & Synch. Primitives**

- **Efficiency Language Compilers**
- **Directed Testing**
- **Dynamic Checking**
- **Debugging with Replay**

- **Static Verification**
- **Type Systems**

- **Correction**
- **OS Libraries & Services**
- **Hypervisor**
- **Legacy OS**
- **Intel Multicore/GPGPU**
- **RAMP Manycore**
Par Lab Research Overview

Easy to write correct software that runs efficiently on manycore

High level, safe, concurrency through high level frameworks

Low level, risky, hardware details fully exposed

Personal Health | Image Retrieval | Hearing, Music | Speech | Parallel Browser

Design Pattern Language (OPL)

Parallel Libraries | Parallel Frameworks

Legacy Code | Schedulers

Communication & Synch. Primitives

Efficiency Language Compilers

OS Libraries & Services

Hypervisor

Intel Multicore/GPGPU | RAMP Manycore

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Correctness

Productivity Layer

Efficiency Layer

Easy to write correct software that runs efficiently on manycore

Applications

Par Lab Research Overview
The Evolution of Frameworks
(with Examples from computational physics)

Domains developed with no interoperability

- linear mechanics
- fluid dynamics
- black hole modeling
- molecular materials

Domains interoperable via wrappers that exchange common data

- linear mechanics
- fluid dynamics
- black hole modeling
- molecular materials

Domain interoperability established through service exchange with common infrastructure

- linear mechanics
- fluid dynamics
- black hole modeling
- molecular materials

Stand alone apps.
Traditionally using a monolithic architecture

Shallow Integration framework ... i.e. the framework doesn’t reach deep inside the components.

Deep Integration framework ... i.e. the framework reaches deep inside the components.
How do we systematically define useful frameworks?

We need a “human language” of Software architecture … Design patterns.

Parallelism as done by traditional HPC* programmers, however, is only part of the problem.

We need patterns that address the overall software architecture

*HPC: High Performance Computing
How do we systematically define useful frameworks?

We need a “human language” of Software architecture ... Design patterns.

Collaboration with Prof. Kurt Keutzer of UC Berkeley (and his brilliant Graduate students)
A Design Pattern Language for Engineering Parallel Applications

Applications

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

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

Concurrent Algorithm Strategy Patterns

Task-Parallelism
Recursive-splitting
Data-Parallelism
Pipeline
Discrete-Event
Geometric-Decomposition
Speculation

Implementation Strategy Patterns

SPMD
Strict-Data-Par.
Program structure
Fork/Join
Actors
Master/Worker
Graph-Partitioning
Loop-Par.
BSP
Task-Queue
Shared-Queue
Shared-Hash-Table
Distributed-Array
Shared-Data
Data structure

Parallel Execution Patterns

MIMD
SIMD
Advancing “program counters”
Thread-Pool
Speculation
Task-Graph
Data-Flow
Digital-Circuits
Message-Passing
Collective-Comm.
Mutual-Exclusion
Point-To-Point-Sync.
Collective-Sync.
Transactional-Mem.
Coordination

Pattern examples

Structural Patterns: Define the software structure .. *Not* what is computed

Computational Patterns: Define the computations “inside the boxes”

Parallel Patterns: Defines parallel algorithms

- Pipe-and-Filter
- Iterative refinement
- MapReduce
- Structured mesh
- Graphical Models
- Fork-join
- SPMD
- Data parallel
A Design Pattern Language for Engineering Parallel Applications

Applications

- **Structural Patterns**
  - Pipe-and-Filter
  - Agent-and-Repository
  - Process-Control
  - Event-Based/Implicit-Invocation
  - Puppeteer

- **Computational Patterns**
  - Graph-Algorithms
  - Dynamic-Programming
  - Dense-Linear-Algebra
  - Sparse-Linear-Algebra
  - Unstructured-Grids
  - Structured-Grids

Patterns travel together … informs framework design (a pathway for cactus is shown here)

- **Task-Parallelism**
  - Parallel Execution Patterns
    - SPMD
    - MIMD
    - Message-Passing
    - Point-To-Point-Sync.
- **Data-Parallelism**
  - Distributed Execution Patterns
    - Multiprocessors (SMP and NUMA)
    - Shared-Queue
    - Message-Passing
    - Collectives-Comm.
  - Data-Structure
    - Distributed-Array
    - Shared-Data

Implementation Strategy Patterns

- **SPMD**
  - Strict-Data-Par.
  - Program structure
- **MIMD**
  - Thread-Pool
  - Task-Graph
  - Data-Flow
  - Digital-Circuits

Discrete-Event

- Geometric-Decomposition

Speculation

- Data structure

Graphical-Models
- Finite-State-Machines
- Backtrack-Branch-and-Bound
- N-Body-Methods
- Circuits
- Spectral-Methods
- Monte-Carlo

LVCSR Software Architecture

Inference Engine

Beam Search Iterations

Active State Computation Steps

Pipe and Filter

MapReduce

Iterative Refinement

Recognition Network

Pipe-and-filter

Graphical Model

Dynamic Programming

LVCSR = Large vocabulary continuous speech recognition.
Speech Recognition Results

- Architecture expressed as a composition of design patterns and implemented as a C++ Framework.
  - Input: Speech audio waveform
  - Output: Recognized word sequences

- Achieved 11x speedup over sequential version
- Allows 3.5x faster than real time recognition
- Our technique is being deployed in a hotline call-center data analytics company
  - Used to search content, track service quality and provide early detection of service issues

Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010
How do we get performance from frameworks?

- SEJITS: Scalable, embedded, just in time specialization
  - Code with a high level language (e.g. Python or Ruby) that is mapped onto a low level, efficiency language (e.g. OpenMP/C or CUDA).
  - SEJITS system to embed optimized kernels specialized at runtime to flatten abstraction overhead and map onto hardware features.

SEJITS comes from Armando Fox’s group at UC Berkeley.
The detailed structure of SEJITS
ASP: A SEJITS for Python

Productivity app

- .py
- f()
- h()
- .c
- cc/ld
- $
- .so

PLL Interp

ASP.py

Specializer

OS/HW

Asp: A SEJITS for Python
https://github.com/shoaibkamil/asp/wiki

Third party names are the property of their owners.
The detailed structure of SEJITS
ASP: A SEJITS for Python

Selective

f()
h()

JIT

cc/ld

.so

Human-expert-authored templates & strategy engine

Embedded

ASP.py

Specialization

Specializer

PLL Interp

OS/HW

Selective

tivity app

Third party names are the property of their owners.

Asp: A SEJITS for Python
https://github.com/shoaibkamil/asp/wiki
Example Code

from stencil_kernel import *

class Laplacian3D(StencilKernel):

    def kernel(self, in_grid, out_grid):
        for x in self.interior_points(out_grid):
            for y in self.neighbors(in_grid, x, 1):
                out_grid[x] += (1.0/6.0) * in_grid[y]

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
from stencil_kernel import *

class Laplacian3D(StencilKernel):
    def kernel(self, in_grid, out_grid):
        for x in self.interior_points(out_grid):
            for y in self.neighbors(in_grid, x, 1):
                out_grid[x] += (1.0/6.0) * in_grid[y]
Introspect to Generate an Abstract Syntax Tree (AST)

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
Transform into an Intermediate Representation (IR)

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
• This is the phase of the infrastructure where the efficiency programmer does most of his/her work
• Use of SEJITS infrastructure for common transformations

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
void kernel_optimized(double* in_grid, double* out_grid) {
    #define min(_a, _b) (_a < _b ? _a : _b)
    #define _idx(d0, d1, d2) (_d2+(_d0 * 258*258)+(_d1 * 258))

    for (int x1x1 = 1; (x1x1 <= 256); x1x1 = (x1x1 + (1 * 192))) {
        for (int x2x2 = 1; (x2x2 <= 256); x2x2 = (x2x2 + (1 * 160))) {
            #pragma omp parallel for
            for (int x1 = x1x1; (x1 <= min((x1x1 + 191), 256)); x1 = (x1 + 1)) {
                for (int x2 = x2x2; (x2 <= min((x2x2 + 159), 256)); x2 = (x2 + 1)) {
                    #pragma ivdep
                    for (int x3 = 1; (x3 <= (256 - 3)); x3 = (x3 + (1 * 4))) {
                        int x4;
                        x4 = _idx(x1, x2, x3);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 - 1), (x2 + 0), (x3 + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 - 1), (x3 + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1))]);
                        x4 = _idx(x1, x2, (x3 + 1));
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 1) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 - 1), (x2 + 0), (x3 + 1) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 1) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 - 1), (x3 + 1) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1) + 1))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1) + 1))]);
                        x4 = _idx(x1, x2, (x3 + 2));
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 2) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 - 1), (x2 + 0), (x3 + 2) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 2) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 - 1), (x3 + 2) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 2) + 1))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 2) + 1))]);
                        x4 = _idx(x1, x2, (x3 + 3));
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 3) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 3) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 3) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 - 1), (x3 + 3) + 0))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3) + 1))]);
                        out_grid[x4] = (out_grid[x4] + (1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3) + 1))]);
                    }
                }
            }
        }
    }
}

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
Updates: Stencil DSEL Performance

~2.5x faster than Pochoir

Geometric mean of 93% of attainable peak.

Source: Shoaib Kamil and Armando Fox, Microsoft Faculty Summit, 2012
How do these two shapes fit together?

Pretty obvious.

How do these two shapes fit together? Not as obvious when dealing with complex, 3D molecular structures.

Why does it matter how molecules fit together? Because most biological processes involve molecular binding.

Henry Gabb: productivity, application programmer
Tim Mattson: specializer writer
Proof-of-Concept Results

• For the productivity programmer:
  • Pattern-based design of application
  • Significantly easier development:
    • Original version: 4,700 lines of C and Perl
    • New version: 500 lines of Python
  • Performance (16-core Intel® Xeon processor):
    • Serial: ~24 hours
    • Parallel: ~3 hours
• For the specializer writer
  • Documentation was a work in progress. Training materials inadequate
  • Error feedback did not track original source code … required a SEJITS expert to find and fix bugs.
  • Assumed specializer writer was a hardcore python programmer (scipy, numpy, etc.).
My Ah-ha moment!!!!

FTDock – Protein Docking

- Independent dockings in 3D search space
- Requires one-line change to application.
- Achieves 290x speedup on 450 cores.

class FtdockMRJob(AspMRJob):
def mapper(self, coords, ignored):
  args = self.data['protein_data']
score = ftdock(*coords, *args)
yield 1, score

FTDock Throughput vs. Problem Size

Source: M. Driscoll, E. Georgana, P. Koanantakool, 2012 ParLab winter Retreat.
Summary

• Many-core processors are here to stay … adapt or become irrelevant.
• “Standards” supported by robust tools that dependably run everywhere is our only hope … but those standards can only flourish if users demand them.
• Long term … we’ll never make everyone an expert at the low level details of mapping parallel algorithms onto hardware.

Solution?
- Separation of concerns .. Productivity vs. efficiency programmers
- A discipline of systematic software architectures defined in terms of a common set of design patterns.
- Software frameworks to turn patterns in to code.
- Software transformation tools to make frameworks fast.