OpenCL

Design Patterns and the Quest for General Purpose Parallel Programming

Tim Mattson

Intel Labs

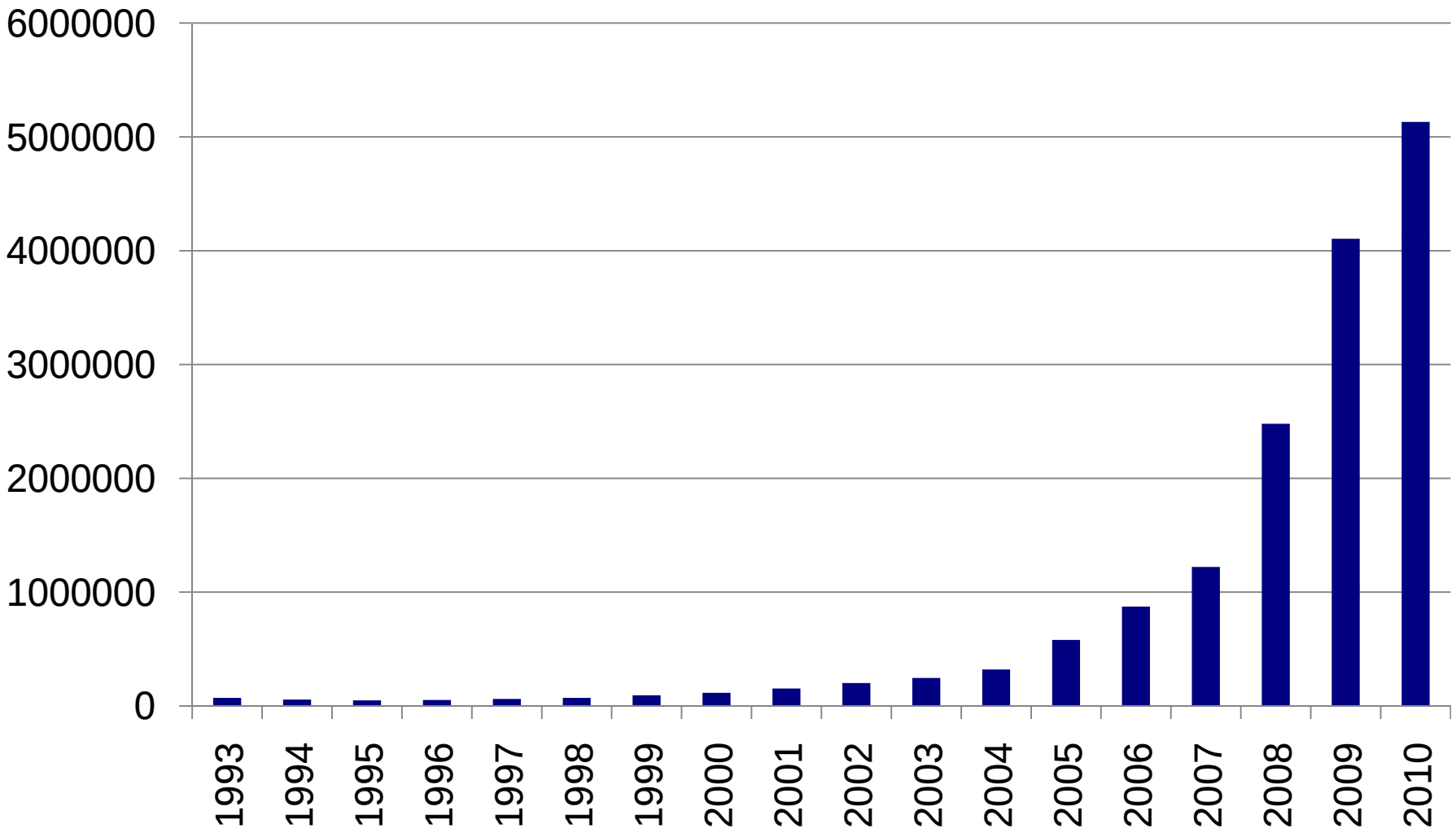
timothy.g.mattson@intel.com

Disclosure

- The views expressed in this talk are those of the speaker and not his employer.
- I am in a research group and know nothing about Intel products. So anything I say about them is highly suspect.
- This was a team effort, but if I say anything really stupid, it's all my fault … don't blame my collaborators.

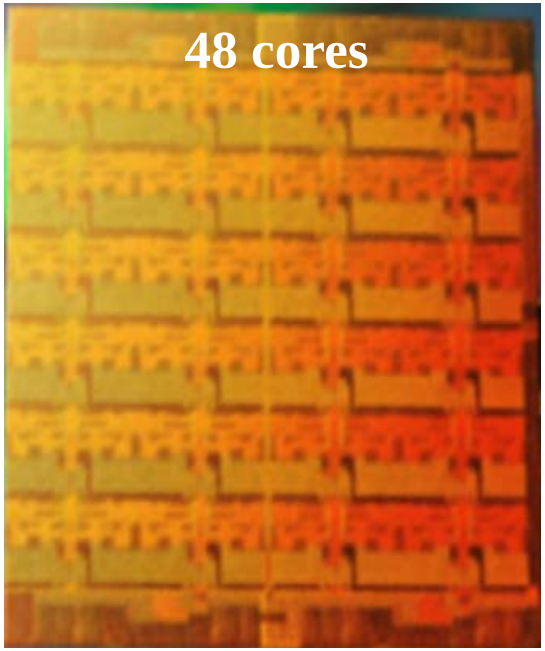
Computing trends ...

Top 500 supercomputers: total number of processors (1993-2010)

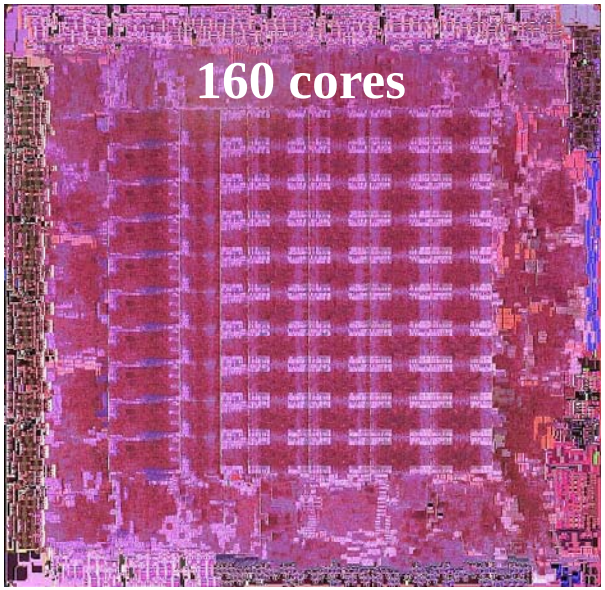


Source: the "June lists" from www.top500.org

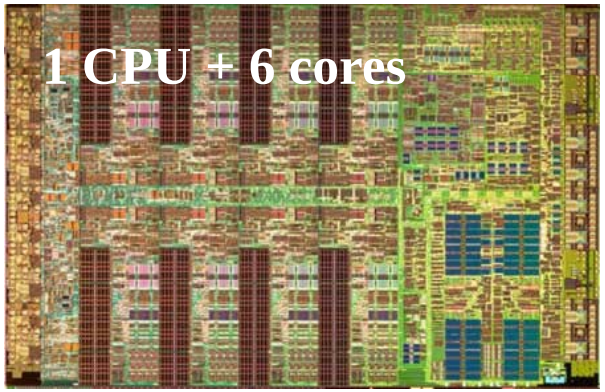
The culprit ... many core chips. And its getting worse!



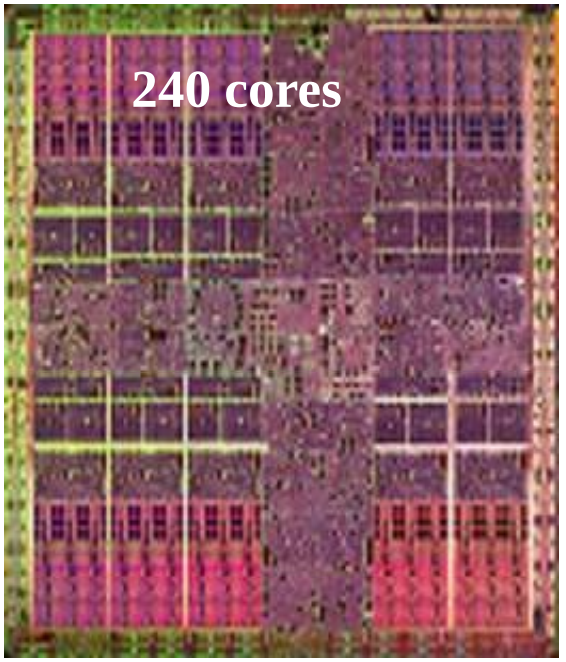
Intel SCC research chip



ATI RV770



IBM Cell

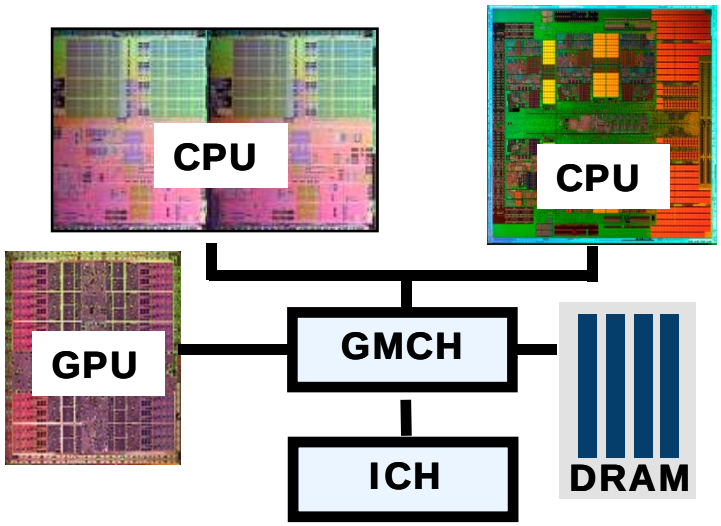


NVIDIA Tesla C1060

3rd party names are the property of their owners.

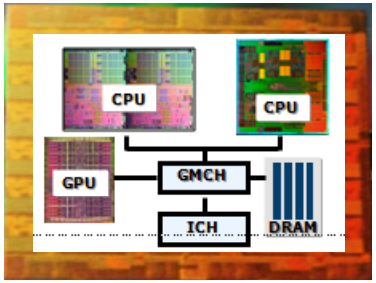
Source: SC09 OpenCL tutorial

Our HW future is clear:



- A modern platform has:
  - CPU(s)
  - GPU(s)
  - DSP processors
  - … other?

- And SOC trends are putting this all onto one chip



The future belongs to heterogeneous, many core SOC as the standard building block of computing

GMCH = graphics memory control hub, ICH = Input/output control hub SOC = system on a chip

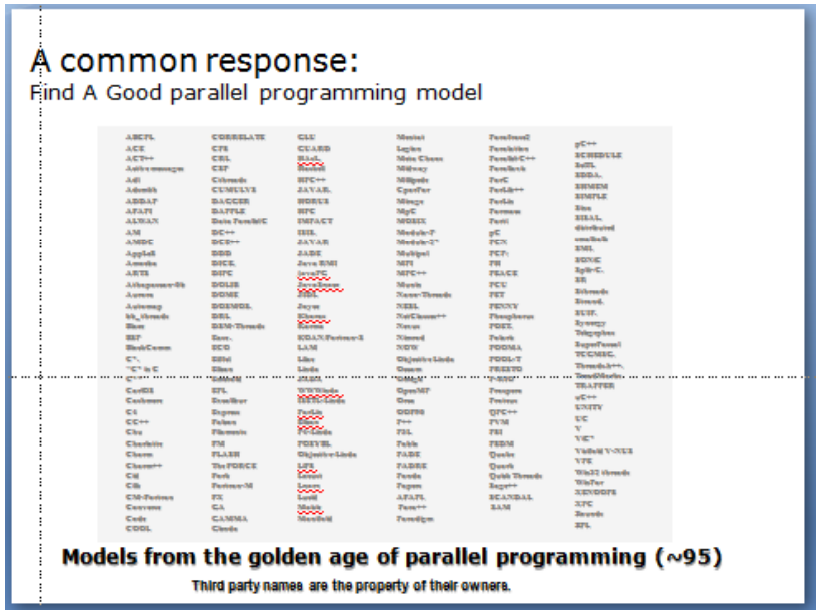
The many core challenge

- 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.

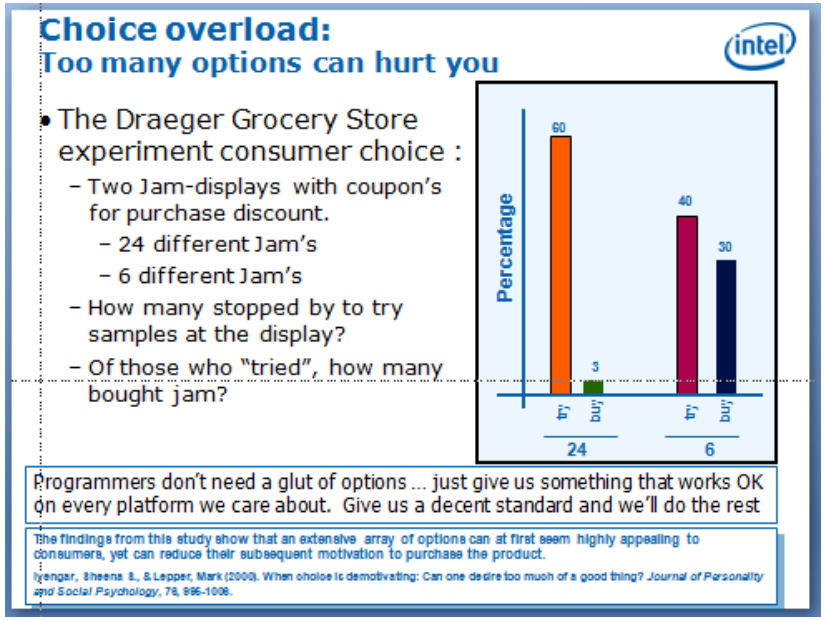
- Result: a fundamental and dangerous (for the computer industry) mismatch
  - Parallel hardware is ubiquitous.
  - Parallel software is rare

- The Many Core challenge ...
  - Parallel software must become as common as parallel hardware
  - Programmers need to make the best use of all the available resources from within a single program:
    - One program that runs close to “hand-tuned” optimal performance) on a heterogeneous platform.

Slides from 2005 …



Leave it to computer scientists and they'll respond to the many core challenge by creating lots of new languages …



… and ISVs respond by running away … choice overload is real

My optimistic view from 2005 ...

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've learned our lesson … we emphasize a small number of industry standards

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.

But we didn't learn our lesson

History is repeating itself!

A small sampling of models from the NEW golden age of parallel programming (2010)

Cilk++	Chapel	Go
CnC	Charm++	Hadoop
Ct	ConcRT	mpc
MYO	CUDA	UPC
RCCE	Erlang	PPL
OpenCL	F#	X10
TBB	Fortress	PLINQ

We've lost our way and have slipped back into the "just create a new language" mentality.

If language obsession is not the solution, what is?

- Consider an early (and successful) adopter of many core technologies ...The gaming industry:
- Game development ... gross generalizations:
  - Time and money: 1-3 years for 1-20 million dollars.
    - A “blockbuster” game has revenues that rival that from a major Hollywood movie.
  - Major games take teams with 50 to 100 people.
  - Only a quarter of the people are are programmers.

Many-core Coping in the gaming industry

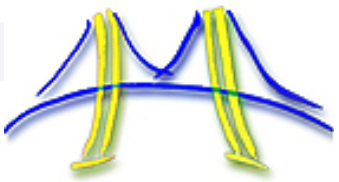
- The key: Enforce a separation of concerns:
  - A small number (<10%) of high priced “Technology programmers” optimize the game engine modules for specific platforms (C, assembly, etc)
  - The rest of the team focuses on the art-work, the story line and putting the basic components together to get the final product “out the door”. (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.

Modular software, frameworks and parallel computing

- As parallelism goes mainstream and reaches extreme levels of scalability ... We can't retrain all the world's programmers to handle disruptive scalable hardware.
- Modular software development techniques and Frameworks will save the day:
  - Framework: A software platform providing partial solutions to a class of problems that can be specialized to solve a specific problem.
- This is not a new idea:
  - Cactus
  - Common Component Architecture
  - SIERRA
  - ... and many others
- ... we just need to push it further and deeper.

We need a systematic way to build a useful collection of frameworks for Scalable computing

Systematic framework design with patterns

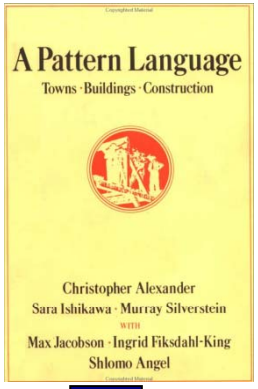


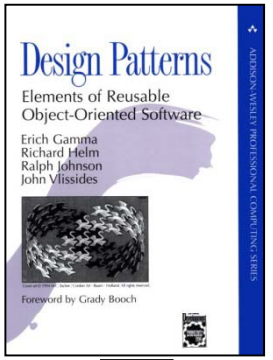
- A **pattern** is a well known solution to a recurring problem ... captures expert knowledge in a form that can be peer-reviewed, refined, and passed-on to others.
- An **architecture** defines the organization and structure of a software system ... it can be defined by a hierarchical composition of **patterns**.
- A **framework** is a software environment based on an **architecture** ... a partial solution for a domain of problems that can be specialized to solve specific problems.

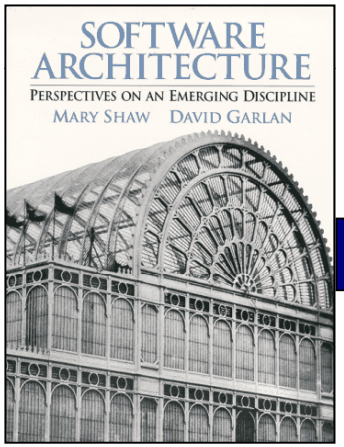
Patterns → Architectures → Frameworks

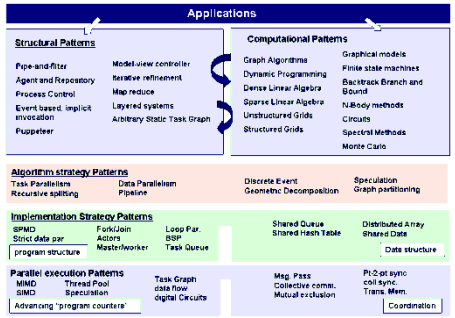
13

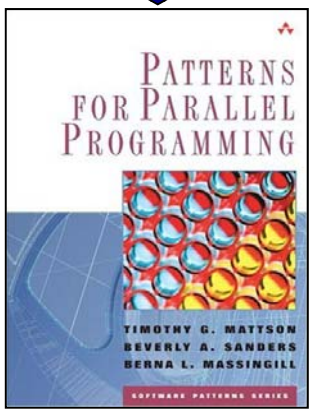
To get frameworks right ... start with an understanding of software architecture



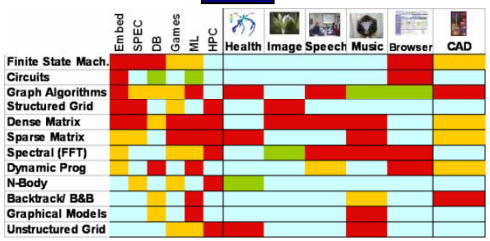








PLPP: Pattern language of Parallel Programming



13 dwarves

Joint work with Kurt Keutzer and his group at UC Berkeley

PA AS

OPL/PLPP 2010

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
- Divide and Conquer
- Data-Parallelism
- Pipeline
- Discrete-Event
- Geometric-Decomposition
- Speculation

Implementation Strategy Patterns

Program structure

- SPMD
- Data-Par/index-
- Fork/Join
- Actors
- Loop-Par.
- Task-Queue

Data structure

- Shared-Queue
- Shared-map
- Partitioned Graph
- Distributed-Array
- Shared-Data

Parallel Execution Patterns

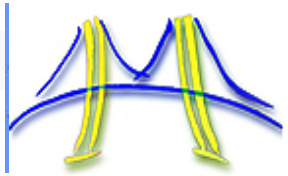
- MIMD
- SIMD
- Thread-Pool
- Task-Graph
- Transactions

Concurrency Foundation constructs (not expressed as patterns)

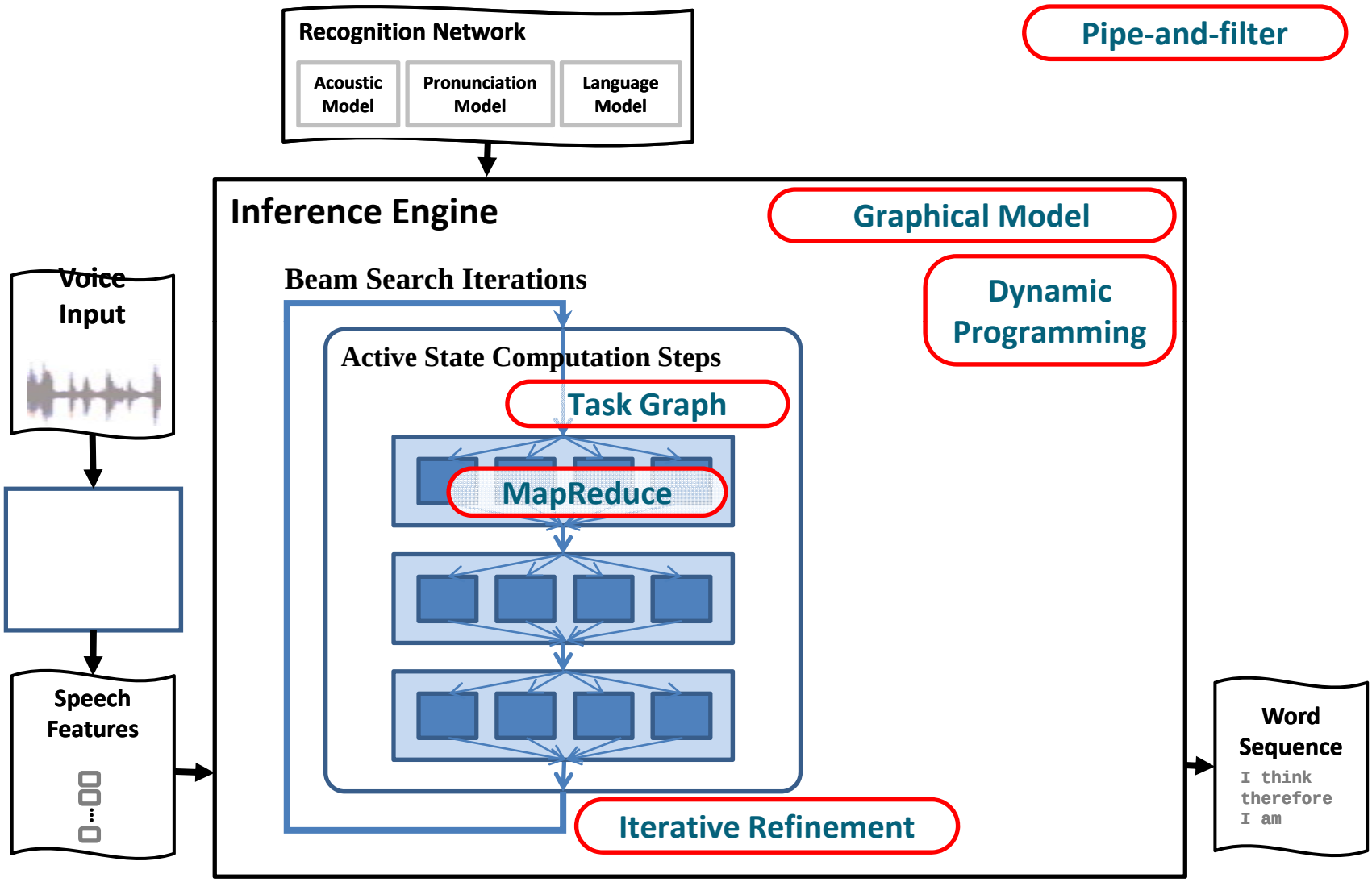
Thread creation/destruction Message-Passing Point-To-Point-Sync. (mutual exclusion)

Pro

http://parlab.eecs.berkeley.edu/wiki/patterns/patterns



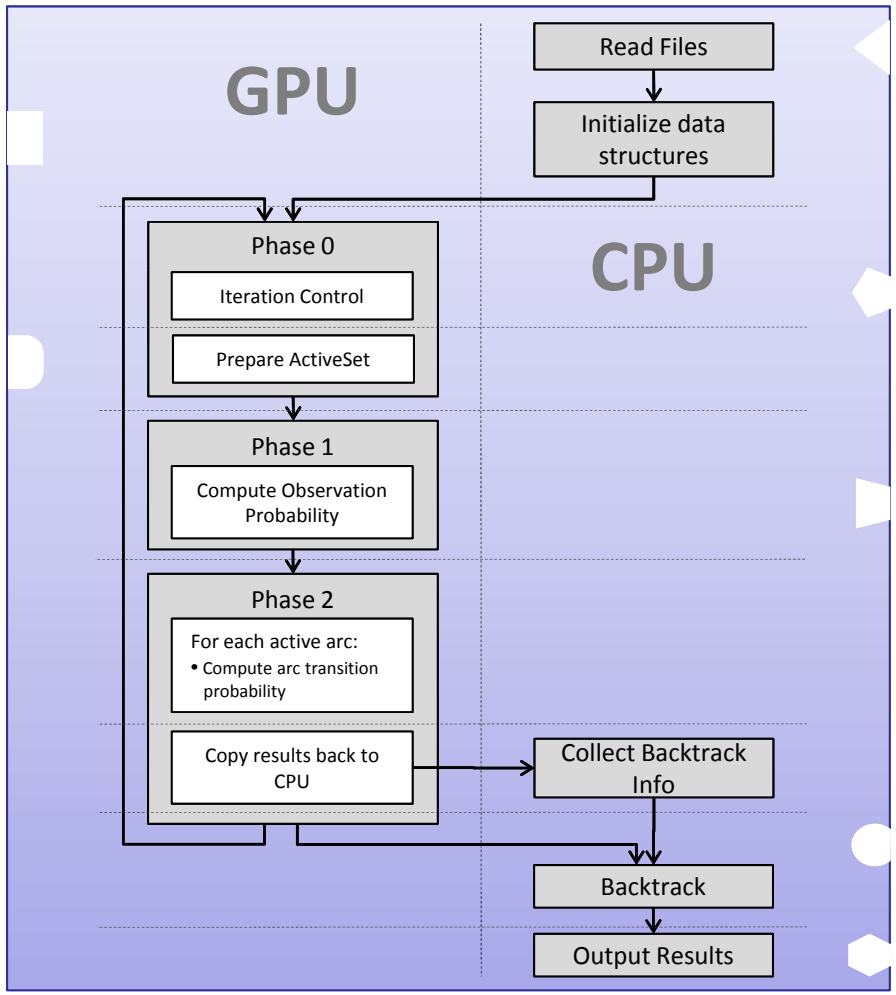
LVCSR Software Architecture



LVCSR = Large vocabulary continuous speech recognition.

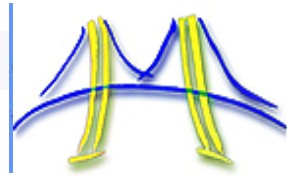
A Speech Framework with Extension

A WFST Based Inference Engine Framework

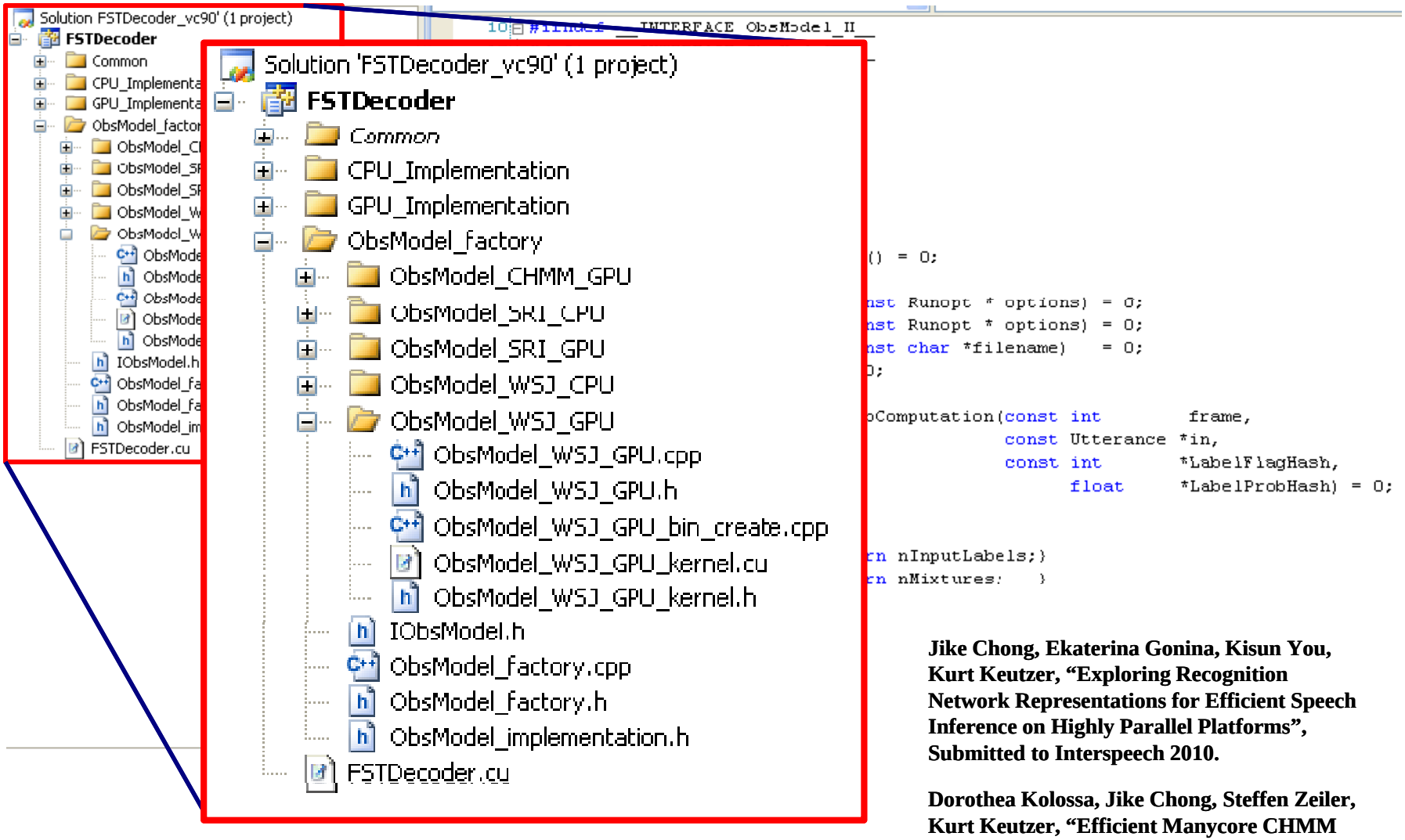


- Two level WFST Network
- One level WFST Network
- Dynamic
- Preload All
- Selective Preload

- HTK Format
- SRI Format
- Fixed Beam Width
- Adaptive Beam Width
- HMM SRI GPU ObsProb
- HMM WSJ GPU ObsProb
- CHMM SRI GPU ObsProb
- Backtrack
- Backtrack + conf. metric
- HResult format
- SRI Scoring format



The Programmer's View



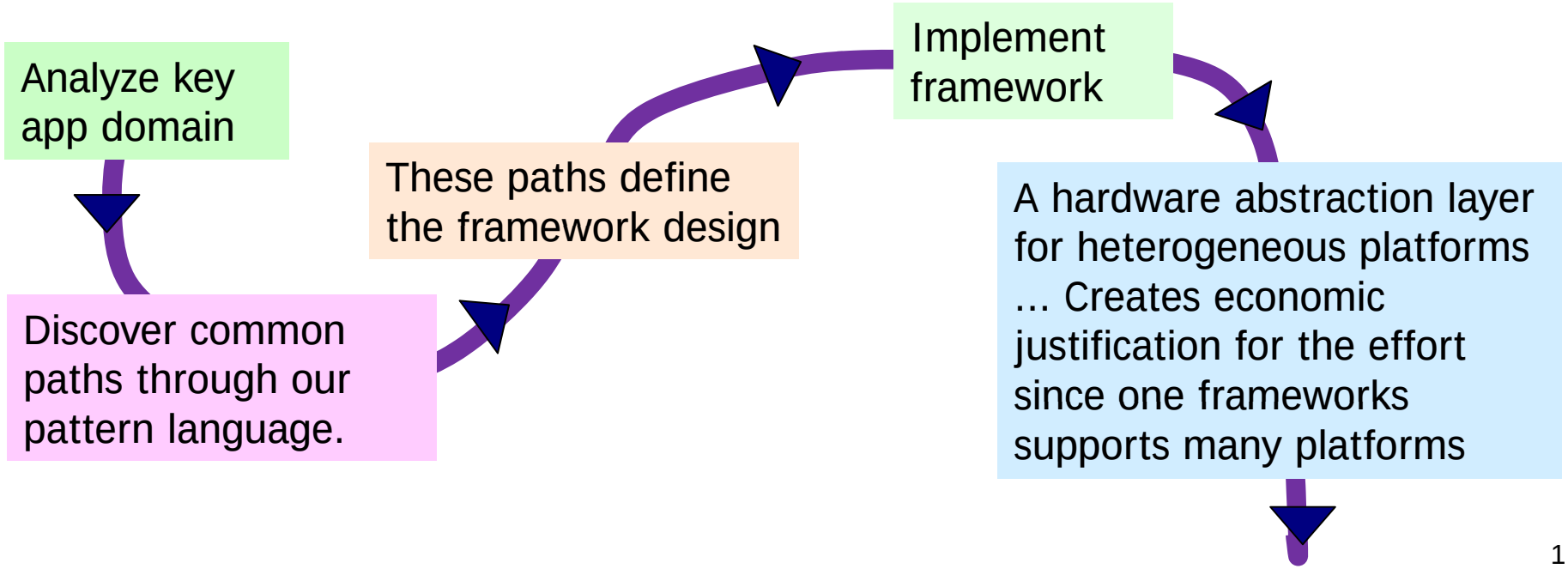
Jike Chong, Ekaterina Gonina, Kisun You, Kurt Keutzer, "Exploring Recognition Network Representations for Efficient Speech Inference on Highly Parallel Platforms", Submitted to Interspeech 2010.

Dorothea Kolossa, Jike Chong, Steffen Zeiler, Kurt Keutzer, "Efficient Manycore CHMM Speech Recognition for Audiovisual and Multistream Data", Submitted to Interspeech 2010.

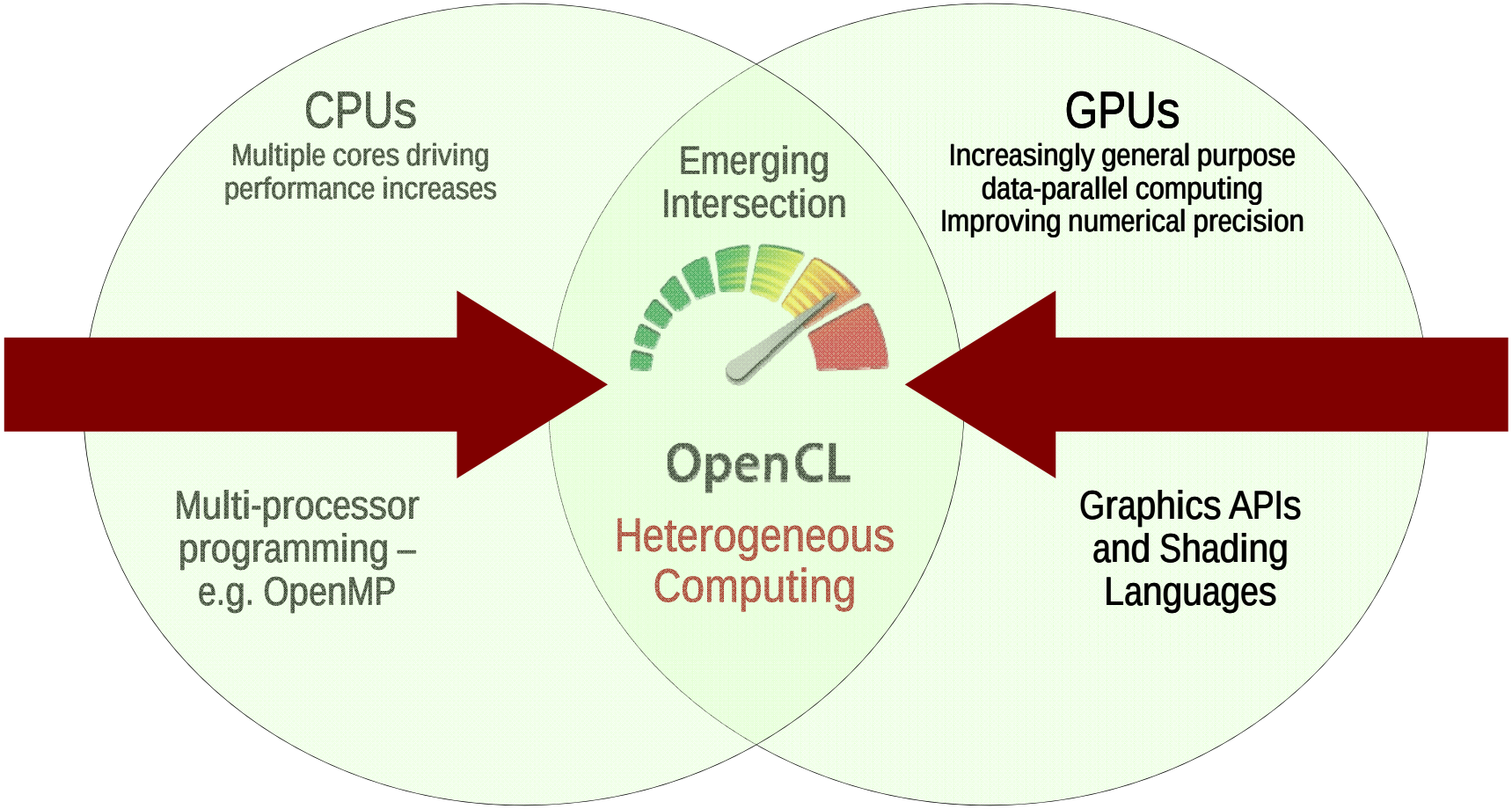
Are Frameworks enough?

- Efficiency ... to support the extremely low overheads required by Amdahl's law ...
  - collapse layers of abstractions. Dynamic optimization.
- A stable, richly supported, and highly portable programming environment to make framework development practical

No time now ... talk to me later about SEJITS



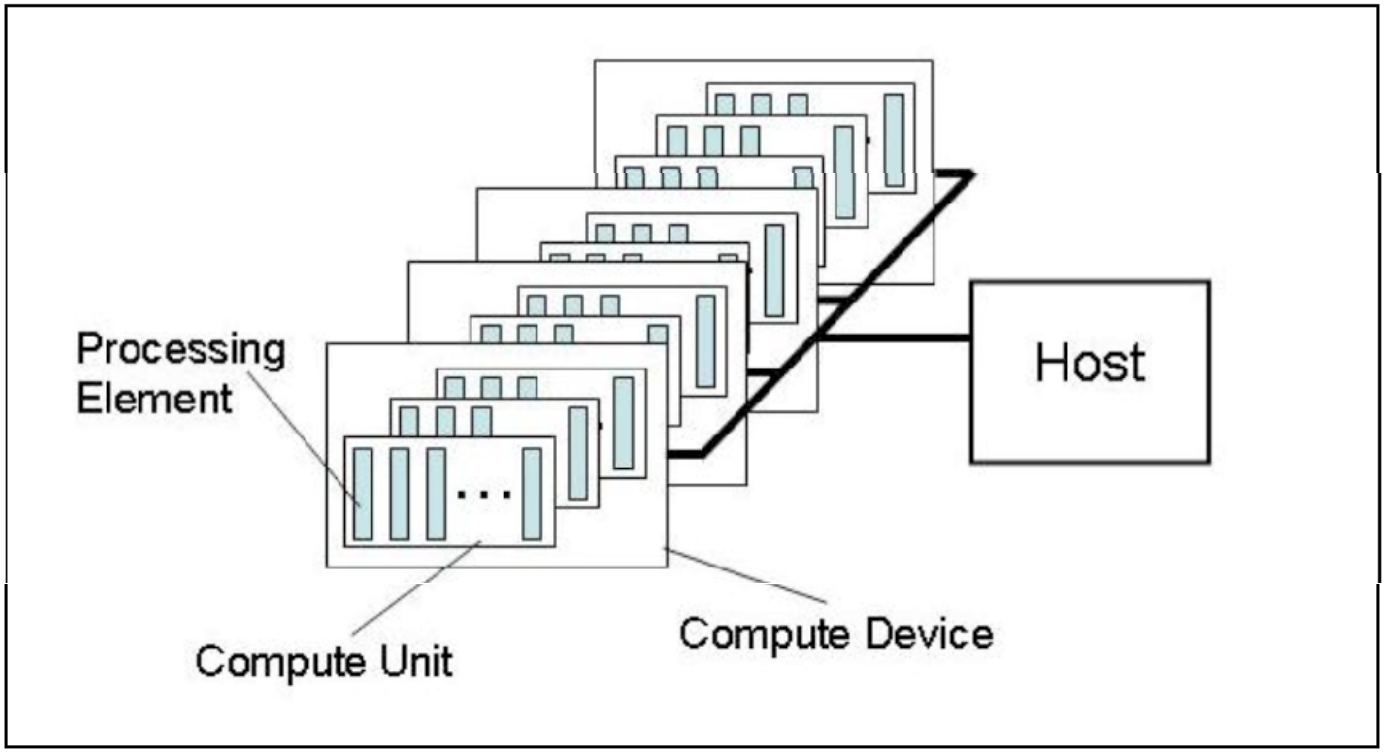
OpenCL: A portable hardware abstraction layer



OpenCL Working Group

- **Diverse industry participation …**
  - **HW vendors (e.g. Apple), system OEMs, middleware vendors, application developers.**
- **OpenCL became an important standard "on release" by virtue of the market coverage of the companies behind it.**

OpenCL Platform Model



- One Host + one or more Compute Devices
  - Each Compute Device is composed of one or more Compute Units
    - Each Compute Unit is further divided into one or more Processing Elements

The BIG idea behind OpenCL

- OpenCL execution model ... define a problem domain and execute a kernel invocation for each point in the domain
  - E.g., process a 1024 x 1024 image: Global problem dimensions: 1024 x 1024 = 1 kernel execution per pixel: 1,048,576 total kernel executions

Traditional loops

```
void
trad_mul(int n,
         const float *a,
         const float *b,
         float *c)
{
  int i;
  for (i=0; i<n; i++)
    c[i] = a[i] * b[i];
}
```

Data Parallel

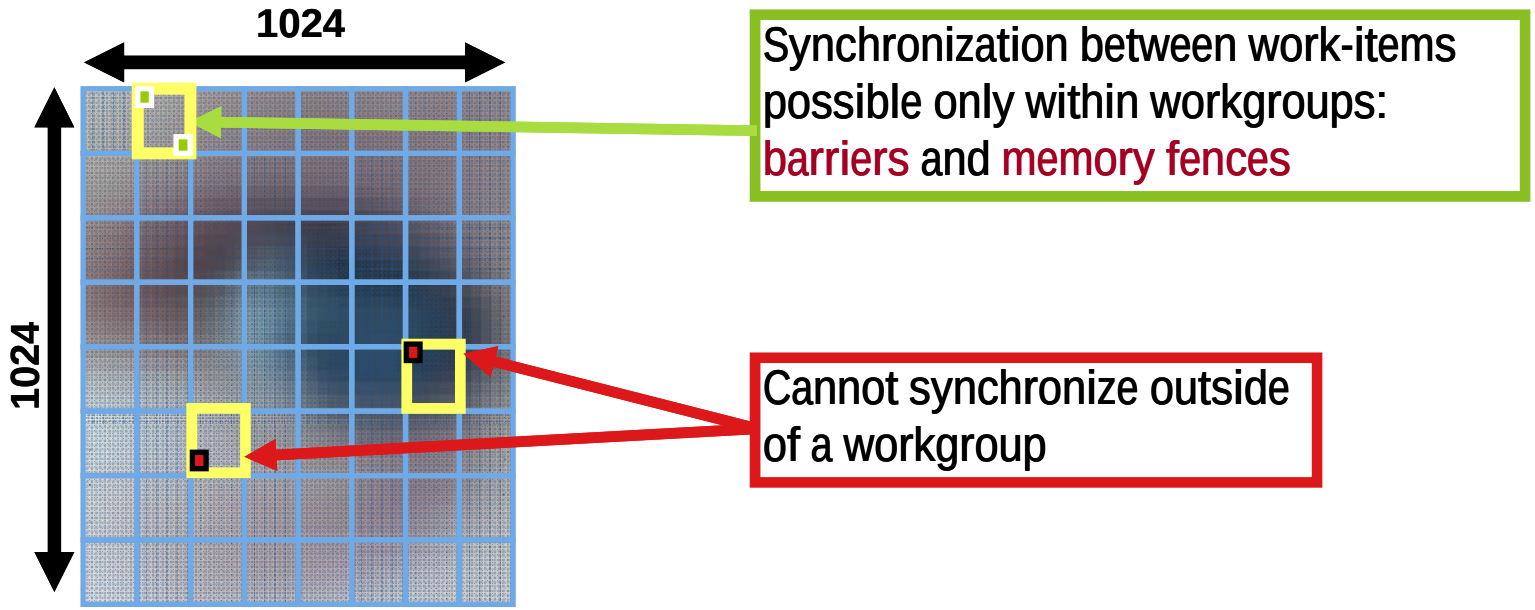
```
kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c)
{
  int id = get_global_id(0);

  c[id] = a[id] * b[id];

} // execute over “n” work-items
```

An N-dimension domain of work-items

- Define an N-dimensioned index space that is “best” for your algorithm
  - Global Dimensions: 1024 x 1024 (whole problem space)
  - Local Dimensions: 128 x 128 (work group … executes together)



Vector Addition: Host Program


```
// create the OpenCL context on a GPU device
cl_context = clCreateContextFromType(0,
   CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);

// get the list of GPU devices associated with
   context
clGetContextInfo(context, CL_CONTEXT_DEVICES, 0,
                                        NULL, &cb);
devices = malloc(cb);
clGetContextInfo(context, CL_CONTEXT_DEVICES, cb,
   devices, NULL);

// create a command-queue
cmd_queue = clCreateCommandQueue(context,
   devices[0], 0, NULL);

// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context,
   CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
   sizeof(cl_float)*n, srcA,
                                        NULL);}
memobjs[1] = clCreateBuffer(context,CL_MEM_READ_ONLY
   | CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcB,
                                        NULL);
memobjs[2] =
   clCreateBuffer(context,CL_MEM_WRITE_ONLY,
                          sizeof(cl_float)*n,
   NULL,
                                        NULL);
// create the program
program = clCreateProgramWithSource(context, 1,
   &program_source, NULL, NULL);
```

```
// build the program
err = clBuildProgram(program, 0, NULL, NULL, NULL,
                                              NULL);

// create the kernel
kernel = clCreateKernel(program, "vec_add", NULL);

// set the args values
err  = clSetKernelArg(kernel, 0, (void *) &memobjs[0],
                                 sizeof(cl_mem));
err |= clSetKernelArg(kernel, 1, (void *)&memobjs[1],
                                 sizeof(cl_mem));
err |= clSetKernelArg(kernel, 2, (void *)&memobjs[2],
                                  sizeof(cl_mem));
// set work-item dimensions
global_work_size[0] = n;

// execute kernel
err = clEnqueueNDRangeKernel(cmd_queue, kernel, 1,
   NULL, global_work_size, NULL, 0, NULL, NULL);

// read output array
err = clEnqueueReadBuffer(cmd_queue, memobjs[2],
   CL_TRUE, 0, n*sizeof(cl_float), dst, 0, NULL, NULL);
```

Vector Addition: Host Program

Define platform and queues

```
    devices[0], 0, NULL);

  // allocate the buffer memory objects
```

Define Memory objects

```
                                          TR,

  memobjs[1] = clCreateBuffer(context,CL_MEM_READ_ONLY
      | CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcB,
      NULL);
  memobjs[2] =
      clCreateBuffer(context,CL_MEM_WRITE_ONLY,
                                  sizeof(cl_float)*n
```

Create the program

Build the program

```
/
K                                                    LL);

// set the args values
err  = clSetKernelArg(kernel, 0, (void *) &memobjs[0],
                                  sizeof(cl_mem));
err |= clSetKernelArg(kernel, 1, (void *)&memobjs[1],
                                  sizeof(cl_mem));
err |= clSetKernelArg(kernel, 2, (void *)&memobjs[2],
                                   sizeof(cl_mem));
```

Create and setup kernel

```
// set work-item dimensions
global_work_size[0] = n;

// execu
err = clEnqueueNDRangeKernel(cmd_queue, kernel, 1,
    NULL, global_work_size, NULL, 0, NULL, NULL);
```

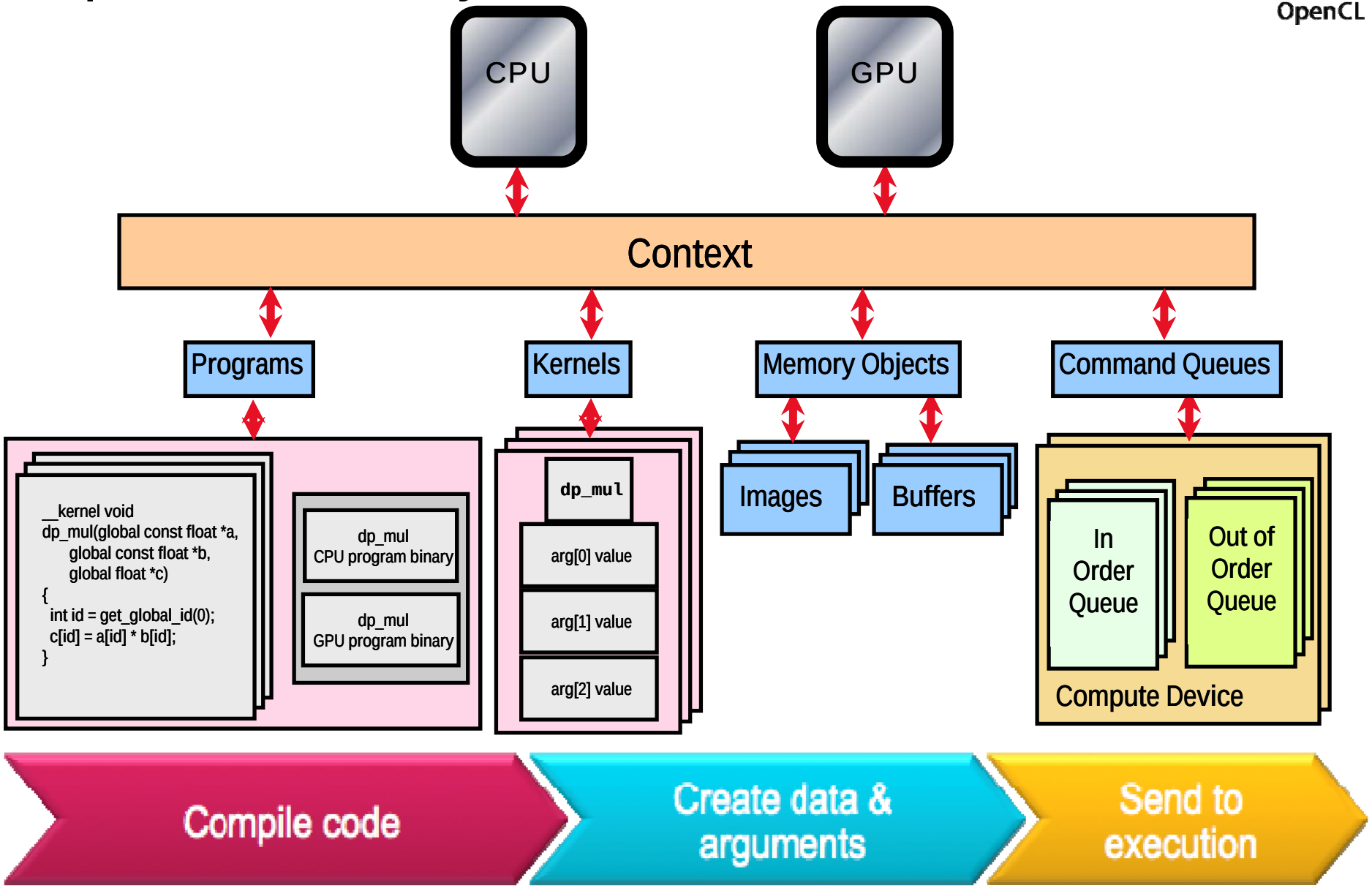
Execute the kernel

```
// rea
err =                                                TRUE,
    0, n*sizeof(cl_float), dst, 0, NULL, NULL);
```

Read results on the host

It's complicated, but most of this is "boilerplate" and not as bad as it looks.

OpenCL summary



What should we do?

- Parallel Programming is hard … so programmers want to do it once … write once and compile to run everywhere.

- We should foster the emergence of a small number of industry standard solutions:
  - OpenMP for shared memory
  - MPI for message passing
  - OpenCL for Heterogeneous computing

- We've lost sight of how incredibly hard it is to create a cross-platform industry standard.

If we respect our programmers, we will keep this in mind and work to make the right standards work, and only introduce new languages as a last resort.

Conclusions

- Many core processors mean all software must be parallel.
- We can't turn everyone into parallel algorithm experts ... we have to support a separation of concerns:
  - Hardcore experts build programming frameworks ... emphasize efficiency using industry standard languages.
  - Domain expert programmers assembly applications within a framework ... emphasize productivity.
- But we (industry) need help ....

Industry is naturally predisposed to trap customers with proprietary languages.

This hurts us all in the long run … so please, only YOU (our "customers") can save us … refuse ANY proprietary programming environment … demand industry standards and force industry to "do the right thing".

Acknowledgements

- Kurt Keutzer (UCB), Ralph Johnson (UIUC) and our community of patterns writers:
  - *Hugo Andrade, Chris Batten, Eric Battenberg, Hovig Bayandorian, Dai Bui, Bryan Catanzaro, Jike Chong, Enylton Coelho, Katya Gonina, Yunsup Lee, Mark Murphy, Heidi Pan, Kaushik Ravindran, Sayak Ray, Erich Strohmaier, Bor-yiing Su, Narayanan Sundaram, Guogiang Wang, Youngmin Yi., Jeff Anderson-Lee, Joel Jones, Terry Ligocki, and Sam Williams.*
- The development of our pattern language has also received a boost from Par Lab faculty — particularly:
  - *Krste Asanovic, Jim Demmel, and David Patterson.*

Backup slides

- Full size version of the “2005 era” slides I referenced

A common response:

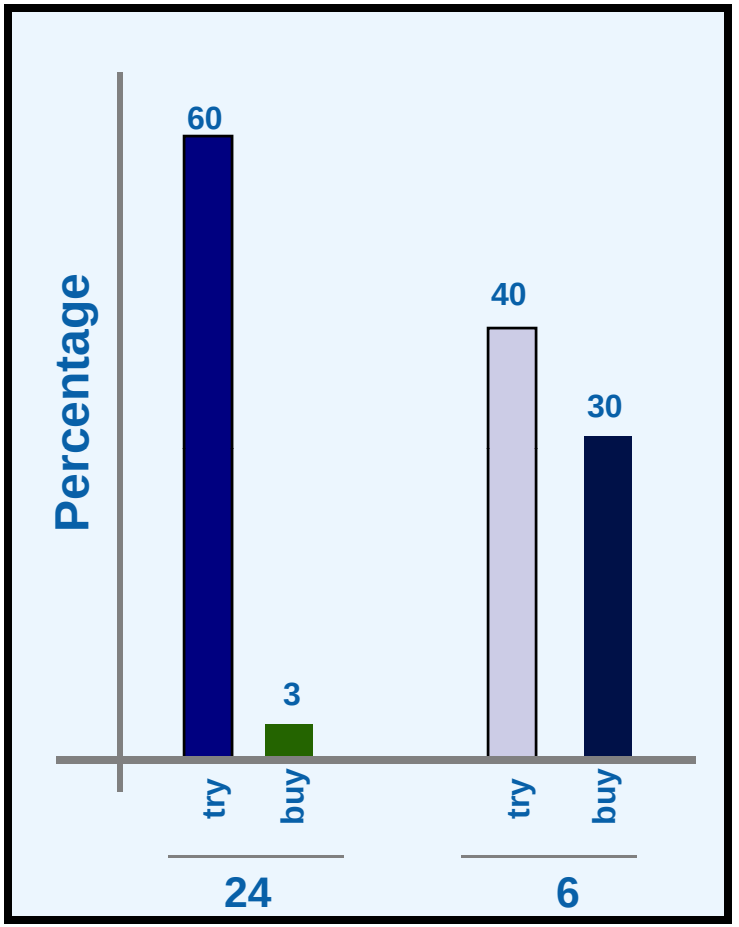
Find A Good parallel programming model

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
CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
ECO
Eiffel
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda
GLU
GUARD
HAsL.
Haskell
HPC++
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold
Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm
Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
PCN
PCP:
PH
PEACE
PCU
PET
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick Threads
Sage++
SCANDAL
SAM
pC++
SCHEDULE
SciTL
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
XENOOPS
XPC
Zounds
ZPL

Models from the golden age of parallel programming (~95)

Choice overload:
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.

Iyengar, Sheena S., & Lepper, Mark (2000). When choice is demotivating: Can one desire too much of a good thing? *Journal of Personality and Social Psychology*, 76, 995-1006.