Parallel Programming:
Can we PLEASE do it right this time?

Tim Mattson
Principal Engineer
Applications Research Laboratory
Intel Corp

1

Abstract

- The computer industry has a problem. As Moore's law marches on, we will be exploiting it to double cores, not frequencies. But all those cores … 2 to 4 today growing to 8, 16 and beyond over the next several years … are of little value with out parallel software. Where will this parallel software come from? With few exceptions, only graduate students and other strange people are willing to write parallel software. Professional software engineers almost never write parallel software.
- Somehow we need to (1) design many core systems programmers can actually use and (2) provide programmers with parallel programming environments that work. The good news is we have 25+ years of history in the HPC space to guide us. The bad news is I don't think very many people are paying attention to these past experiences.
- In this talk, I look back at the history of parallel computing and develop a set of rules we must follow if we want to create many core systems that are actually useful. A common theme is that just about every stupid mistake we could make has already been made by someone. So rather than reinvent these mistakes on our own, lets learn from the past and "do it right this time".

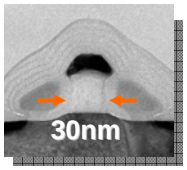
2

Disclaimer

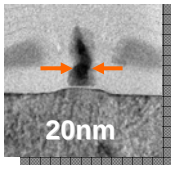
- The views expressed in this presentation are my own and do not represent the views of the Intel Corporation (or its lawyers).

3

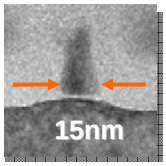
Moore's Law is Going strong



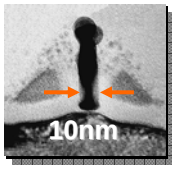
65nm process
2005



45nm process
(20 nm Prototype)
2007



32nm process
(15 nm prototype)
2009



22nm process
(10 nm prototype)
2011

… combined with advanced packaging, we get the familiar transistor-doubling with each generation

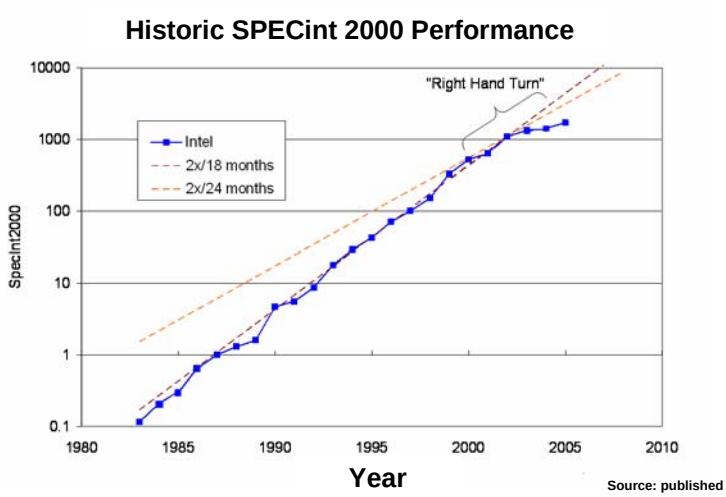
Technology Node (nm)	90	65	45	32	22
Integration Capacity (BT)	2	4	8	16	32

4

These are projections only and may not be reflected in future products from Intel Corp.

Source: Intel

But … Single thread performance is falling off

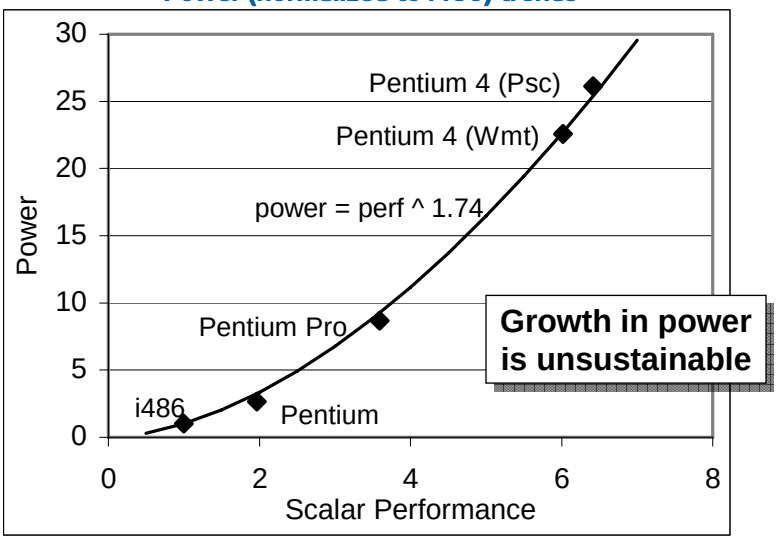


Source: published SPECInt data

5

And power is out of control:

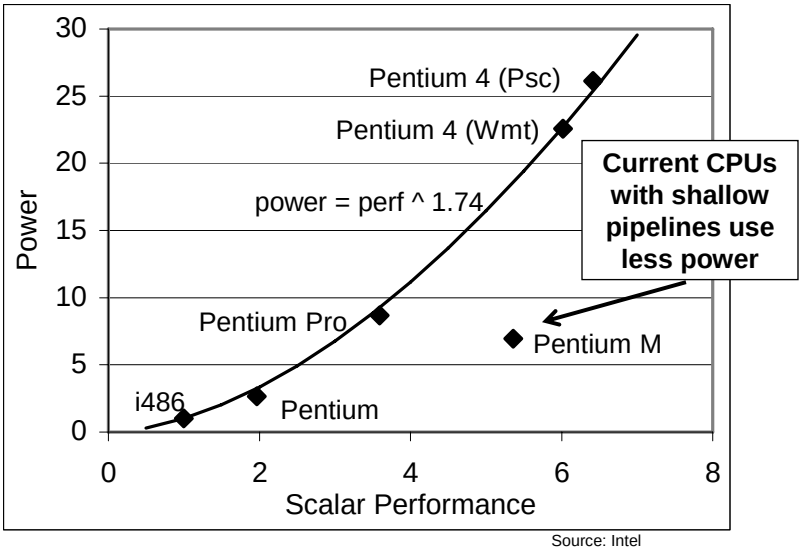
Power (normalized to i486) trends



Source: Intel

6

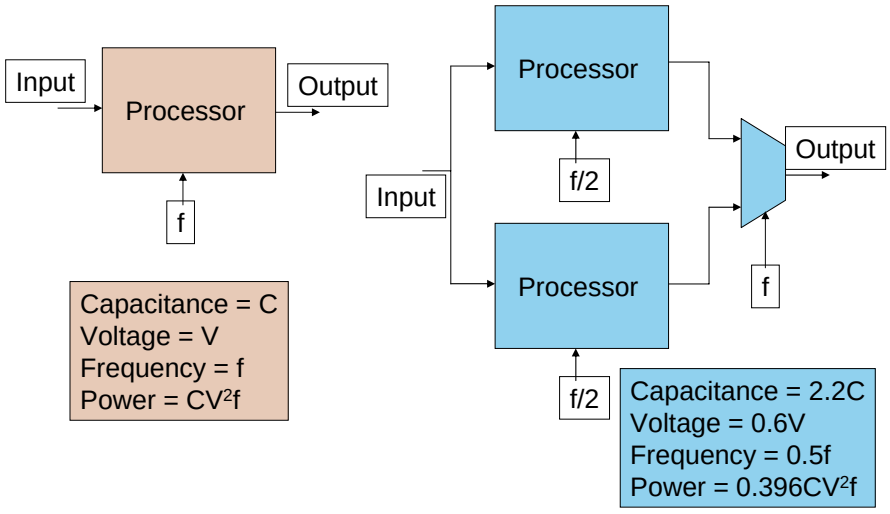
Architecture optimized for power: a big step in the right direction



Source: Intel

7

Using Multiple cores to reduce power even more

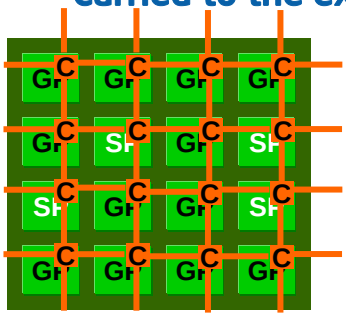


8

Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W., "Optimizing power using transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*,, vol.14, no.1, pp.12-31, Jan 1995

Source: K. Keutzer of UCB

Carried to the extreme … a many core future



General Purpose, Low Power Cores

Special Purpose HW

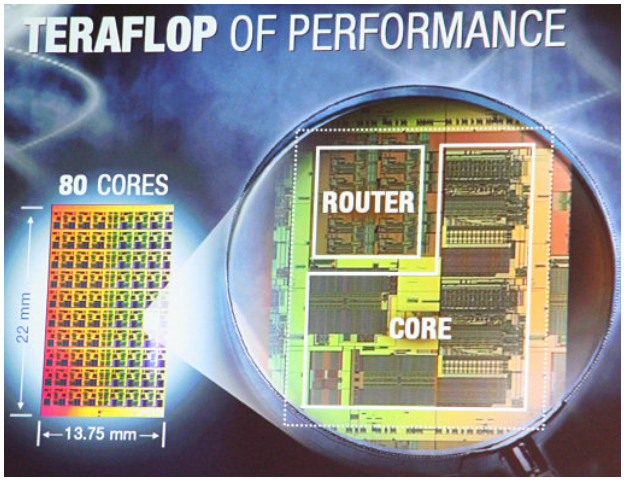
Scalable on-die network

Heterogeneous Multi-Core Platform … we're all doing it (Intel and our competitors)

This is an architecture concept that may or may not be reflected in future products from Intel Corp.

9

We've made good progress with the hardware: Intel's 80 core test chip (2007)



- Performance @ 4.27 GHz and 97 Watts
  - Peak
    1.37 SP TFLOPS
  - Explicit PDE solver
    1.0 SP TFLOPS
  - Matrix Multiply
    0.51 SP TFLOPS
- See Backup slides for more details about this project

This is an architecture concept that may or may not be reflected in future products from Intel Corp.

10

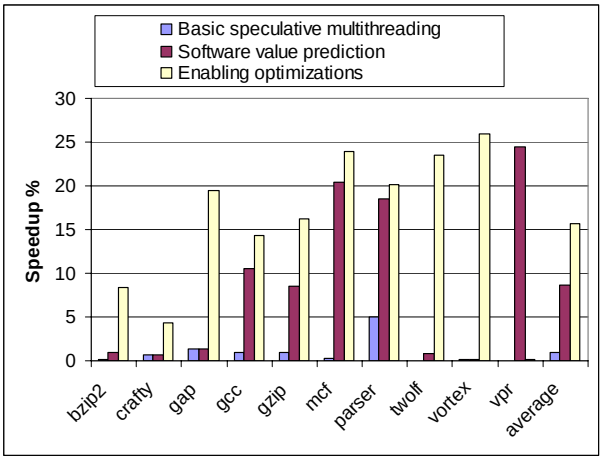
Source: A 80-tile 1.28 TFLOP Network-on-Chip in 65 nm CMOS, ISSCC'07, Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James tschanz, David Finan, Priya Iyer, Arvind Singh, Riju Jacob, Shailendra Jain, Sriram venkataraman, Yatin Hoskote and Nitin Borkar.

Software?

- Many core systems are useless without software that can exploit available concurrency.
- Can we generate parallel software automatically?

11

How about 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 belive Auto-par will solve our problems.

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

12

A Cost-Driven Compilation Framework for Speculative Parallelization of Sequential Programs, Zhao-Hui Du, Chu-Cheow Lim, Xiao-Feng Li, Chen Yang, Qingyu Zhao, Tin-Fook Ngai (Intel Corporation) in PLDI 2004

Software?

- Our only hope is to get programmers to write parallel software "by hand".

  Making this happen is the famous "Parallel programming problem".
- And after 25+ years of research, we are no closer to solving the parallel programming problem …

  Only a tiny fraction of programmers write parallel code.
- Will the "if you build it they will come" principle apply?
  - Many hope so, but ..

    that implies that people didn't really try hard enough over the last 25 years. Does that really make sense?

13

What went wrong during the MPP* era?

Can we do it right this time with multi-core?

Those who cannot remember the past are condemned to repeat it.

(George Santayana 1863-1952)

The lesson's from history:

Rules every parallel programming Environment designer should follow

14

*MPP Massively Parallel Processor

Rule 1:

- It is far better to have a small number of good technologies and build on what is in use today rather than create something completely different and new.

15

All you need is a good Parallel Programming Language, right?

Parallel Programming environments in the 90's

ABCPL	CORRELATE	GLU	Mentat	Parafrase2	pC++
ACE	CPS	GUARD	Legion	Paralation	SCHEDULE
ACT++	CRL	HAsL.	Meta Chaos	Parallel-C++	SciTL
Active messages	CSP	Haskell	Midway	Parallaxis	POET
Adl	Cthreads	HPC++	Millipede	ParC	SDDA.
Adsmith	CUMULVS	JAVAR.	CparPar	ParLib++	SHMEM
ADDAP	DAGGER	HORUS	Mirage	ParLin	SIMPLE
AFAPI	DAPPLE	HPC	MpC	Parmacs	Sina
ALWAN	Data Parallel C	IMPACT	MOSIX	Parti	SISAL.
AM	DC++	ISIS.	Modula-P	pC	distributed smalltalk
AMDC	DCE++	JAVAR	Modula-2*	pC++	SMI.
AppLeS	DDD	JADE	Multipol	PCN	SONiC
Amoeba	DICE.	Java RMI	MPI	PCP:	Split-C.
ARTS	DIPC	javaPG	MPC++	PH	SR
Athapascan-0b	DOLIB	JavaSpace	Munin	PEACE	

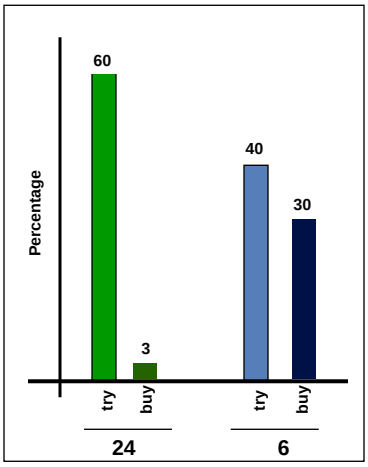
This glut of parallel programming languages actually hurt the cause of parallel computing.

C4	Express	ParLin	OOF90	Prospero	UNITY
CC++	Falcon	Eilean	P++	Proteus	UC
Chu	Filaments	P4-Linda	P3L	QPC++	V
Charlotte	FM	Glenda	p4-Linda	PVM	ViC*
Charm	FLASH	POSYBL	Pablo	PSI	Visifold V-NUS
Charm++	The FORCE	Objective-Linda	PADE	PSDM	VPE
Cid	Fork	LiPS	PADRE	Quake	Win32 threads
Cilk	Fortran-M	Locust	Panda	Quark	WinPar
CM-Fortran	FX	Lparx	Papers	Quick Threads	WWWinda
Converse	GA	Lucid	AFAPI.	Sage++	XENOOPS
Code	GAMMA	Maisie	Para++	SCANDAL	XPC
COOL	Glenda	Manifold	Paradigm	SAM	Zounds
					ZPL

16

So is it really bad to have so many languages?

- 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?



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.

17

Parallel Programming API's today

- Choice overload*:
  - A glut of options scares consumers (i.e. ISVs) away … Less is More
- Today's major APIs
  - 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.

18

*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.

An example of fixing existing APIs :

OpenMP 2.5 can't deal with a simple pointer following loop

```
nodeptr list, p;

For (p=list; p!=NULL; p=p->next)
        process(p->data);
```

OpenMP 3.0 fixes this by adding a new task construct:

```
nodeptr list, p;

#pragma omp parallel
{
    #pragma omp single
    {
        for (p=list; p!=NULL; p=p->next)
        #pragma omp task firstprivate(p)
            process(p->data);
    }
}
```

19

Rule 2:

- When developing parallel programming technologies, work with production level programs produced by application programmers

20

Rule 2 example: HPF

- HPF (High Performance Fortran) was created in the early 90's by academics to solve important data parallel problems.
- Worked great on toy problems
- Was terrible for real applications. Why? They picked the wrong programming model.
  - Even a data parallel algorithm includes task parallel components so HPF was a nightmare to apply to real applications

Successful technologies that followed rule 2 include OpenMP, PVM, MPI, and TCGMSG.

The GPGPU inspired rush to resurrect SIMD style programming models is an example of a modern violation of this rule.

21

Rule 3:

- If the goal is to build a commercially relevant market, industry must be heavily involved … if not the driver.

22

Rule 3 example: MPI 2

- MPI 2.0 was defined by the MPI forum, a group dominated at that time by national lab and academic researchers
- Specification completed in April 1997, generally available conforming implementations, Nov 2004* (MPIch2)
  - Academics are fundamentally interested in research agendas, not building markets!

OpenMP and MPI 1.0 did it right and had implementations ready when the specs were released.

23

* This is the date MPIch2 was released. It is true that Pallas did an implementation of MPI2 for Fujitsu in or around 2002. But this was not available across the industry.

Rule 4:

- Work on the really important problems, not just your favorite problems.

24

Software Issues for many core: Top 10 list

1. Finding concurrent tasks in a program. How to help programmers "think parallel"?
2. Scheduling tasks at the right granularity onto the processors of a parallel machine
3. The data locality problem: Associating data with tasks and doing it in a way that our target audience will be able to use correctly.
4. Supporting scalability: hardware - bandwidth and latencies to memory plus interconnects between processors to support applications that scale.
5. Supporting scalability: software – libraries, scalable algorithms, and adaptive runtimes to map high level software onto low l platform details.
6. Synchronization constructs (and protocols) that let programmers write programs free from deadlock and race conditions that scale across the full system.
7. Tools, API's and methodologies to support the debugging process
8. Error recovery and support for fault tolerance
9. Support for good software engineering practices: composability, incremental parallelism, and code reuse.
10. Support for portable performance. What are the right models (or abstractions) so programmers can write code once and expect it to execute well on the parallel platforms we care about in the market.

25

Paul Petersen, Arch Robison,
Bruce Leasure, Tim Mattson

But it seems everyone is focused on transactional memory

- Transactional memory only addresses a few of the lower priority problems.
  6. Synchronization constructs (and protocols) that let our target programmers write programs that are free from deadlock and race conditions but still scale across the full system.
  8. Error recovery and support for fault tolerance
  9. Support for good software engineering practices: composability, incremental parallelism, and code reuse.
- TM will help, but it won't be the "game changer" some have promised.
  - R&D resources are roughly fixed … by putting so much into TM, we are putting less into the more pressing problems.

26

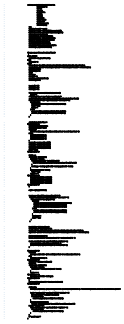
Rule 5:

- We must stop acting like engineers or worse, "marketing hacks".
  - Engineers: think it, build it, demo it, declare victory.
  - Marketers focus on "cheap pot shots".
- We can only solve the parallel programming problem by systematic, scientific methodologies.
  - Scientist: Think it, hypothesize, build it, test hypothesis, build a theory, iterate as needed to develop a science of parallel programming.

27

Rule 5 example: Sparse matrix vector product

OpenMP



172 lines of code

Nested Data Parallel

```
VEC<double> sparseMatrixVectorProduct(
  VEC<double> A, VEC<int> rowindex,
  VEC<int> cols, VEC<double> v)
{
  VEC expv = distribute(v,cols);
  VEC product = A*expv;
  return multiReduceSum(product,rowindex);
}
```

6 lines of code
Better performance & scalability

Nested data parallelism enables safe and scalable composition of software modules

- What conceptually does this comparison tell us? Anything?
- Isn't this just marketing-speak disguised as reasoned analysis?

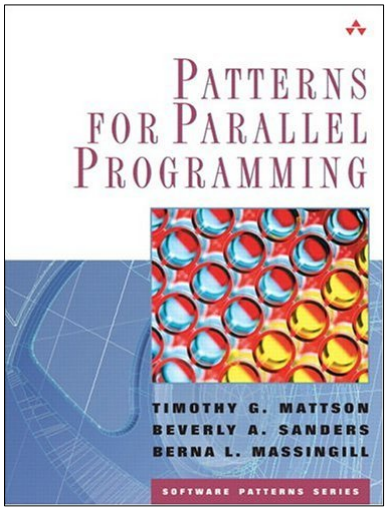
28

Let's do it right this time.

- Let's stop taking "pot shots" at each others APIs and adopt a more disciplined scientific approach:
- Science is a community process … if we want to make progress on programmability, we need to:
  - Develop a systematic, Human-centered model of how programmers solve parallel programs
  - Define a human-language of programmability … so we can objectively discuss pros-and-cons of different programming technologies.
  - Define metrics so we can track progress and make systematic comparisons between APIs

29

A model of how parallel programmers think



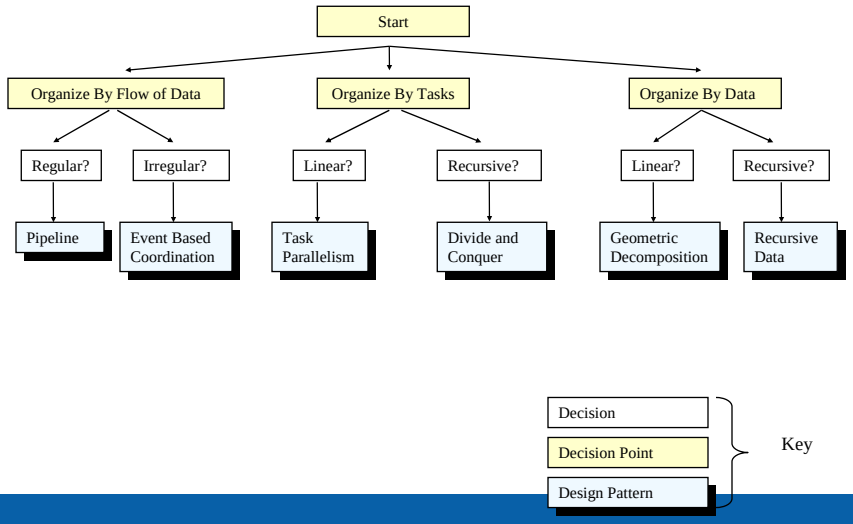
A design pattern language for parallel algorithm design with examples in MPI, OpenMP and Java.

This is our hypothesis for how programmers think about parallel programming.

NOTE: this is just a hypothesis … a starting point. It needs more peer review and experiments to validate our theories.

30

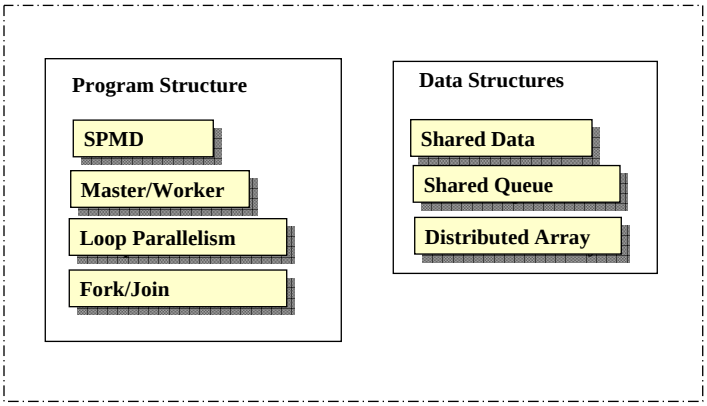
The AlgorithmStructure Design Space



31

The Supporting Structures Design Space

High level constructs impacting large scale organization of the source code.



32

Let's do it right this time.

- Let's stop taking "pot shots" at each others APIs and adopt a more disciplined scientific approach:
- Science is a community process … if we want to make progress on programmability, we need to:
  - Develop a systematic, Human-centered model of how programmers solve parallel programs
  - Define a human-language of programmability … so we can objectively discuss pros-and-cons of different programming technologies.
  - Define metrics so we can track progress and make systematic comparisons between APIs

33

A "human" language of programmability

- Thomas Green is a well known researcher in the "psychology of programming" community.
- After years of work on formal cognitive models with little to show for it, he concluded:

  The way forward is not to make strong, simple claims about how cognitive process work. The way forward is to study the details of how notations convey information.

- He proposed a set of "Cognitive Dimensions" as a "discussion framework" for information notations.
- Cognitive Dimensions in action
  - First used to analyze visual programming languages.
  - Since then, its used to analyze a number of information appliances.
  - Used by Steven Clarke of Microsoft to analyze C#

34

Cognitive dimensions

- There are around 13 of them. The 10 most important to parallel programming are:
  - Viscosity: how hard is it to introduce small changes.
  - Hidden Dependencies: does a change in one part of a program cause other parts to change in ways not overtly apparent in the program text?
  - Error Proneness: How easy is it to make mistakes?
  - Progressive Evaluation: can you check a program while incomplete? Can parallelism be added incrementally?
  - Abstraction Gradient: how much is required? How much abstraction is possible
  - Closeness of mapping: how well does the language map onto the problem domain?
  - Premature commitment: Does the notation constrain the order you do things? AKA imposed look ahead.
  - Consistency: Similar semantics implied by similar syntax. Can you guess one part of the notation given other parts?
  - Hard mental operations: does the notation lead you to complex combinations of primitive operations
  - Terseness: how succinct is the language?
- For parallel programming, I'll add two more
  - HW visibility: is a useful cost model exposed to the programmer?
  - Portability: does the notation assume constraints on the hardware?

35

Cognitive Dimensions: viscosity

- How easy is it to introduce changes to an existing parallel program?
- Low viscosity example: To change how loop iterations are scheduled in OpenMP, just change a single clause

```
#pragma omp parallel for reduction(+:sum) private(x) schedule(dyanmic)
        for (i=1;i<= num_steps; i++){
                x = (i-0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step *h sum;
```

36

Cognitive Dimensions: viscosity

- How easy is it to introduce changes to an existing parallel program?
- High viscosity example: To change how loop iterations are scheduled in Win32 threads, change multiple lines of code

```
step = 1.0/(double) num_steps;
for (i=start;i<= num_steps; i=i+NUM_THREADS){
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
}
EnterCriticalSection(&hUpdateMutex);
global_sum += sum;
LeaveCriticalSection(&hUpdateMutex);
}
```

```
step = 1.0/(double) num_steps;
Initiallize_task_queue(num_steps);
while(!done){
    I = get_next()
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
    done = termination_test(i);
}
EnterCriticalSection(&hUpdateMutex);
global_sum += sum;
LeaveCriticalSection(&hUpdateMutex);
}
```

37

Cognitive Dimensions: Error Proneness

- Shared address space languages such as OpenMP are very error prone.
- Consider this simple program:

```
#include <omp.h>
static long num_steps = 100000;        double step;
void main ()
{              int i;       double x, pi, sum = 0.0;
               step = 1.0/(double) num_steps;
#pragma omp parallel for reduction(+:sum)
               for (i=1;i<= num_steps; i++){
                              x = (i-0.5)*step;
                              sum = sum + 4.0/(1.0+x*x);
               }
               pi = step * sum;
}
```

By forgetting a simple "private(x)" clause, I've introduced a race condition

38

Cognitive Dimensions: Abstraction depth

- Abstraction: make the issues you care about visible, hide the rest.
  - Abstraction rich parallel languages:
    - TBB (thread building blocks); generic programming and standard template libraries meets parallel programming.
      - Build abstract containers, and introduce parallelism by using concurrent containers
      - Change how concurrency is executed by changing containers.
  - Abstraction poor languages:
    - OpenMP: Programmer has very little support from the notation for building abstractions. Very little abstraction is possible.
- Abstraction barriers: how much abstraction is required just to get started.
  - TBB has a very high abstraction barrier.

39

Cognitive Dimensions: Hidden dependencies

- Hidden Dependencies: make a change in one location and effects seen elsewhere … in ways not apparent in the program text.
- Abstraction rich languages increase problems from hidden dependencies:
  - Change member function in a base class, and an object of a derived class changes its behavior.

40

Cognitive dimensions of OpenMP and MPI

Cognitive Dimension	OpenMP	MPI
Viscosity	Low viscosity: pragma have minimal semantic weight … easy to move around	High viscosity: sends/recvs paired, data structures explicitly decomposed
Error Proneness	High: shared address space = hard to detect race conditions	Medium-low: disjoint memory makes races rare and deadlock easy to find. Long argument lists are a problem.
HW visibility	Poor: An abstract API that hides hardware	Fair to Good: hardware model implied but usually visible.
Progressive evaluation	High: Semantically neutral constructs allow incremental parallelism.	Low: rip prog. apart to expose distributed data and tasks, and test once you put things back together.
Portability	Poor: requires systems with shared address spaces	Great: assumes minimal system support

41

Let's do it right this time.

- Let's stop taking "pot shots" at each others APIs and adopt a more disciplined scientific approach:
- Science is a community process … if we want to make progress on programmability, we need to:
  - Develop a systematic, Human-centered model of how programmers solve parallel programs
  - Define a human-language of programmability … so we can objectively discuss pros-and-cons of different programming technologies.
  - Define metrics so we can track progress and make systematic comparisons between APIs

42

Metrics of programmability

- We have benchmarks for performance, how about for programmability?
  - HPCS took a stab at the problem with their synthetic compact applications, but they didn't take it far enough.
  - The old Salishan problems were great, but need updating.
    - Hamming's Problem (compute ordered sets of prime numbers): recursive streams with producer/consumer parallelism and recursive tasks.
    - The Paraffin Problem: nested loop-level parallelism over complex tree structures
    - The doctor's office: asynchronous processes with circular dependencies
    - Skyline matrix solver: solving structure sparse problems.

43

A programmability benchmark suite

- Let's define a set of programmability benchmarks.
  - The key is coverage … we must cover the major classes of applications and parallel algorithms.
- The programmability benchmarks must be:
  - Provided as serial code in a common high level language.
  - Contain lots of concurrency; accessible but not too easily.
  - Have a "right" answer that can be easily verified.
  - Short … you want users to focus on the parallel notation, not the program itself.

44

A programmability benchmark suite

- The famous "view from Berkeley" paper defined thirteen dwarfs … common clusters of algorithm/application classes:

Dense Linear Alg.	Sparse Lin. Alg.	Structured grids
Unstruc. grids	Spectral methods	N-body methods
Dynamic prog	Back-track/branch and bound	MapReduce
Graph traversal	Graphical methods	Combinatorial logic
	Finite state mach.	

- We could create the "13 exemplars" … i.e. one instance from each cluster.
- But the best approach would be for "end user" communities to tell us what to do.
  1. Professional societies from CAD, gaming, business IT, etc. could offer their top three programmability benchmarks.
  2. We'd remove overlap and end up with a converged set of relevant benchmarks.

45

So what can you do?

- If you are a parallel computing researcher:
  - Act like a scientist, generate hypothesis, conduct experiments, and avoid the stupid mistakes of the past.
  - Build off existing APIs (as IBM did with X10). Create new ones only as a last resort.
- If you are a user (i.e. use software others write for you)
  - Start insisting on parallel application software
- If you are a software developer
  - Start engineering parallel code … and demand mature languages with well developed tool chains.
  - Avoid people selling short-cuts based on grand promises, magic and fantasy.
- If you are a computer vendor … the problems are bigger than any one of us. We need to work together to enable serious parallel software engineering.
  - We can use the OpenMP architecture review board as a model for competitors working together to build the standards we all need.

46

Conclusion

- Many core hardware is progressing nicely.
- Many core software is stuck … lots of good ideas chasing good problems but no framework to support systematic progress.
- Solution: Let's do it right this time:
  - Learn from past work in parallel computing … fix old languages before creating new ones.
  - Keep centered on the human-side of programming: A community accepted design pattern language defining standard practice in parallel algorithm design.
  - Act like scientists with peer review and a well defined language of programmability.
  - Metrics to drive real solutions: Standard programmability benchmarks.

47