in an era of Ubiquitous Parallel Computing

James Reinders, Director, Evangelist, Intel Software
james.r.reinders@intel.com

Tuesday September 29, 09:15- 10:30

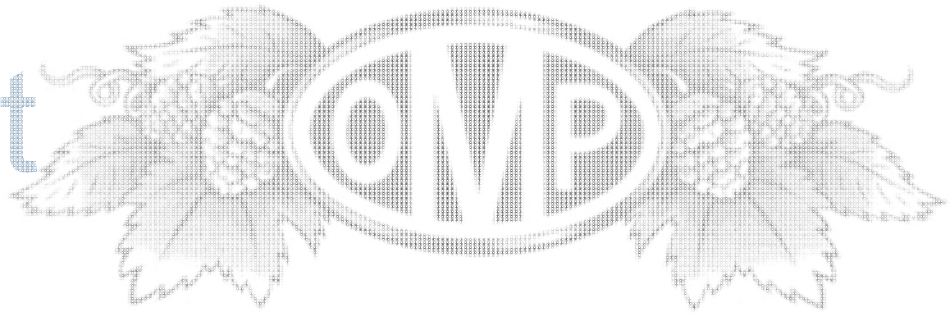
James has been fortunate to work on a many parallel computing projects and books. As a featured speaker, he will share thoughts and observations primarily drawn from recent book projects which have involved scores of contributors and most frequently featured OpenMP. James observes that we have survived a decade of “no more free lunch” and find parallelism is everywhere. OpenMP certainly rules supreme in HPC. The stark reality is that OpenMP is needed more than ever, which is quite an accomplishment for a model introduced during the “free lunch” era.

I ♥ OpenMP

C$OMP DO IT IN PARALLEL

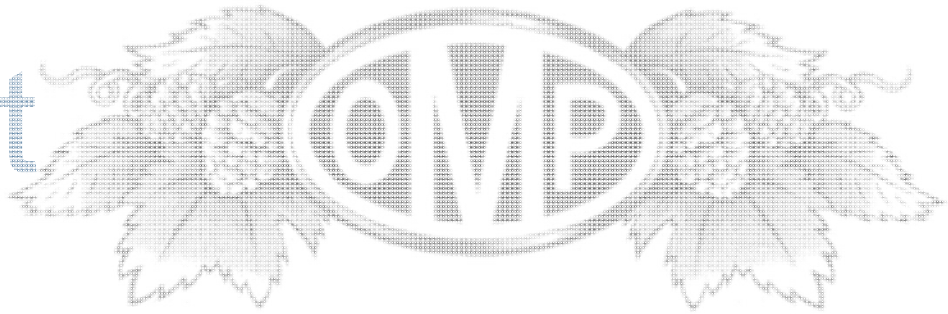
OpenMP* Past

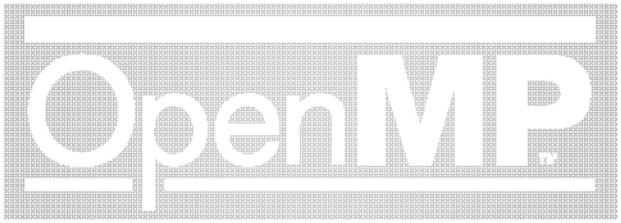
OpenMP Past



OpenMP* Present

OpenMP Past





OpenMP Present

OpenMP* Future

OpenMP* Past

OpenMP* Past

– 2014

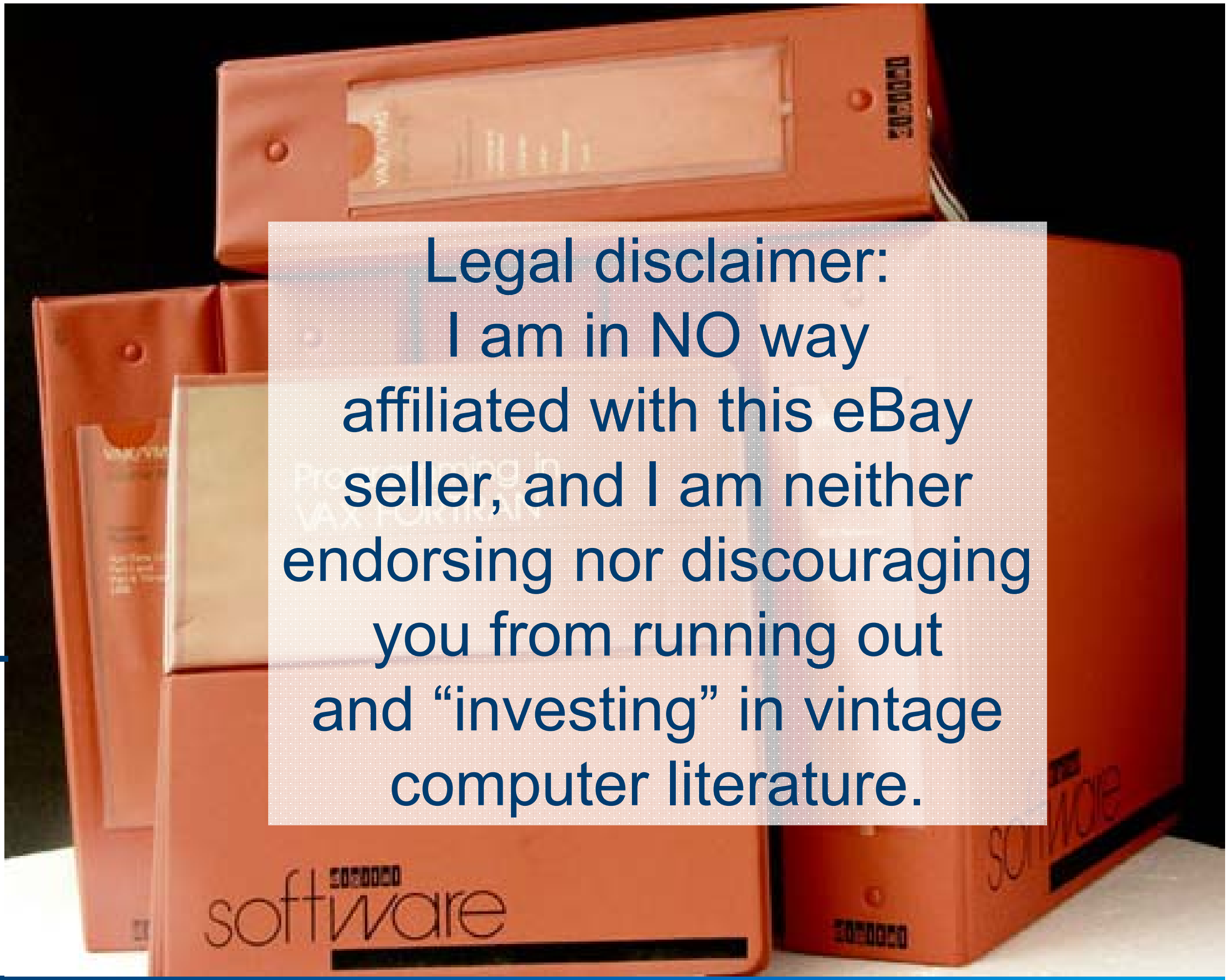
c. 1995
We love directives,
that’s why we have so many

I wish I had taken a photo of my desk.

It would have included a scene like this.

By the way… on eBay now US$835 + shipping US$149.

“RARE DIGITAL EQUIPMENT VAX / VMS Manuals…”



Legal disclaimer:
I am in NO way affiliated with this eBay seller, and I am neither endorsing nor discouraging you from running out and “investing” in vintage computer literature.

Seriously...
in 1995,
I had DEC,
Sun, Cray,
IBM, KAI,
Intel, etc.
manuals.,
because
I led the
Intel Fortran
compiler team.

so OpenMP was born

Fortran Oct. 1997

C/C++ Oct. 1998

Pop Quiz

63, 148, 240, 250, 326, 354, 320, …

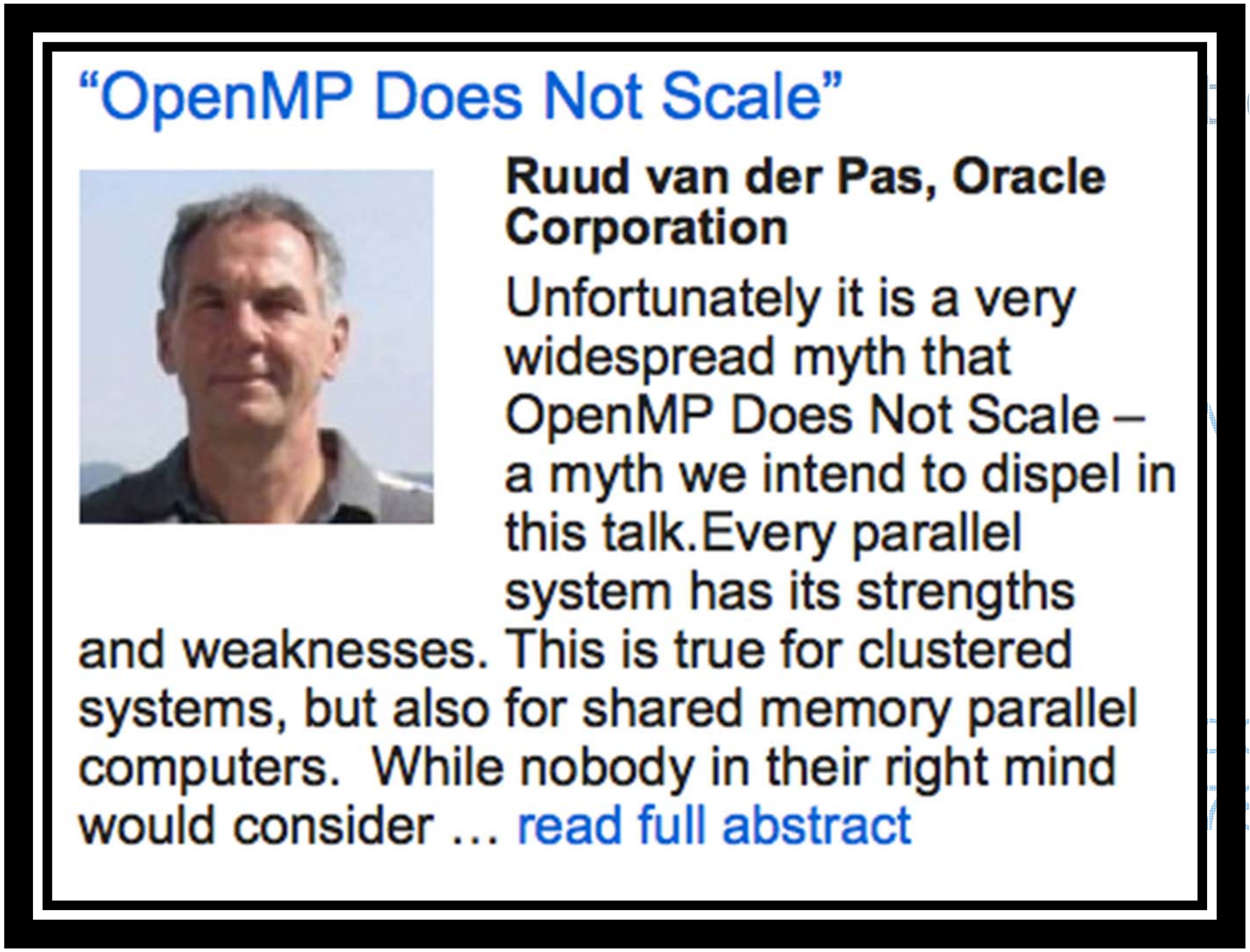
What is the *next* number in this sequence?

Pages in the OpenMP specifications

(as retrieved on September 27, 2015)

63 (1.0 Fortran, Oct 1997),
148 (1.0, C/C++ & Fortran 63, Oct 1998),
240 (2.0, C++ 106 + Fortran 124, March 2002),
250 (2.5, May 2005),
326 (3.0, May 2008),
354 (3.1, July 2011),
320 (4.0, July 2013)*** examples doc is 258 pages (320+258 = 578)

Next is ??? ☺



“OpenMP Does Not Scale”

Ruud van der Pas, Oracle Corporation

Unfortunately it is a very widespread myth that OpenMP Does Not Scale – a myth we intend to dispel in this talk.Every parallel system has its strengths and weaknesses. This is true for clustered systems, but also for shared memory parallel computers. While nobody in their right mind would consider … read full abstract

Ruud,
thank you for
the inspiration.

(I’m not
blaming him
for my humor.)

ons

998),

arch 2002),

oc is 258 pages

8)

OpenMP* Present

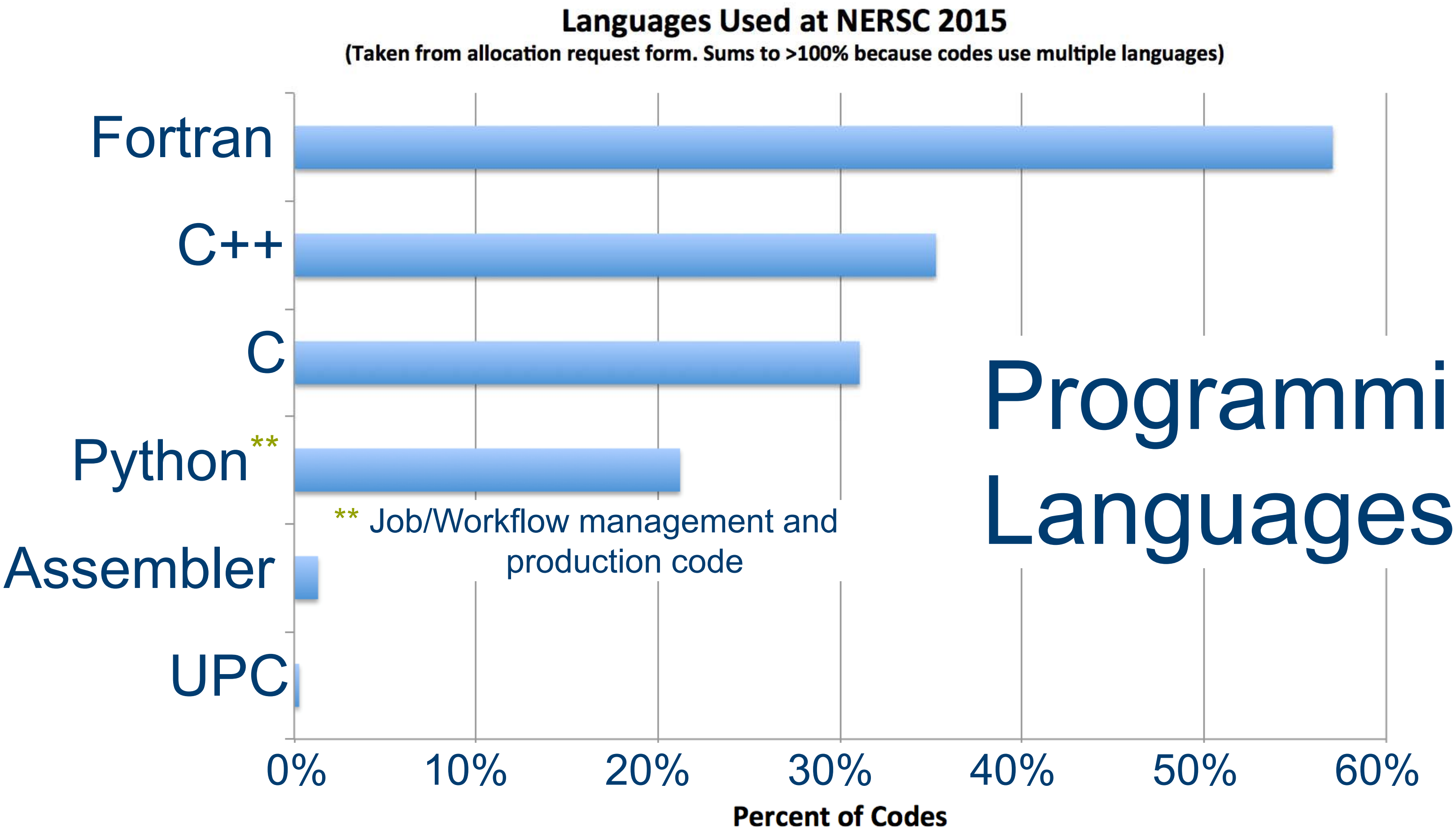
OpenMP* Present

2015

OpenMP certainly rules supreme in HPC.

OpenMP is needed more than ever, which is quite an accomplishment for a model introduced during the “free lunch” era.

Programming Languages

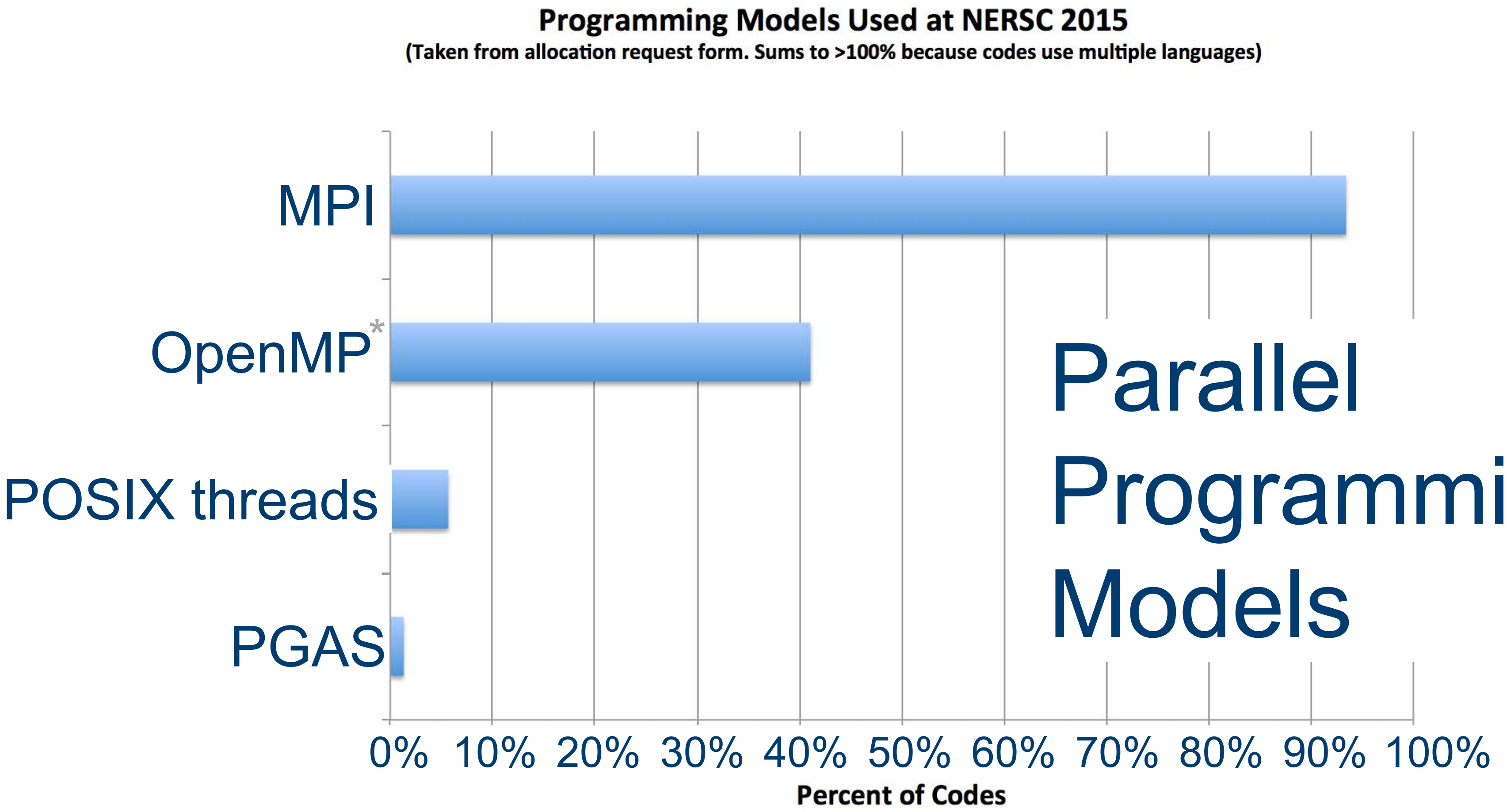


Languages Used at NERSC 2015

(Taken from allocation request form. Sums to >100% because codes use multiple languages)

** Job/Workflow management and production code

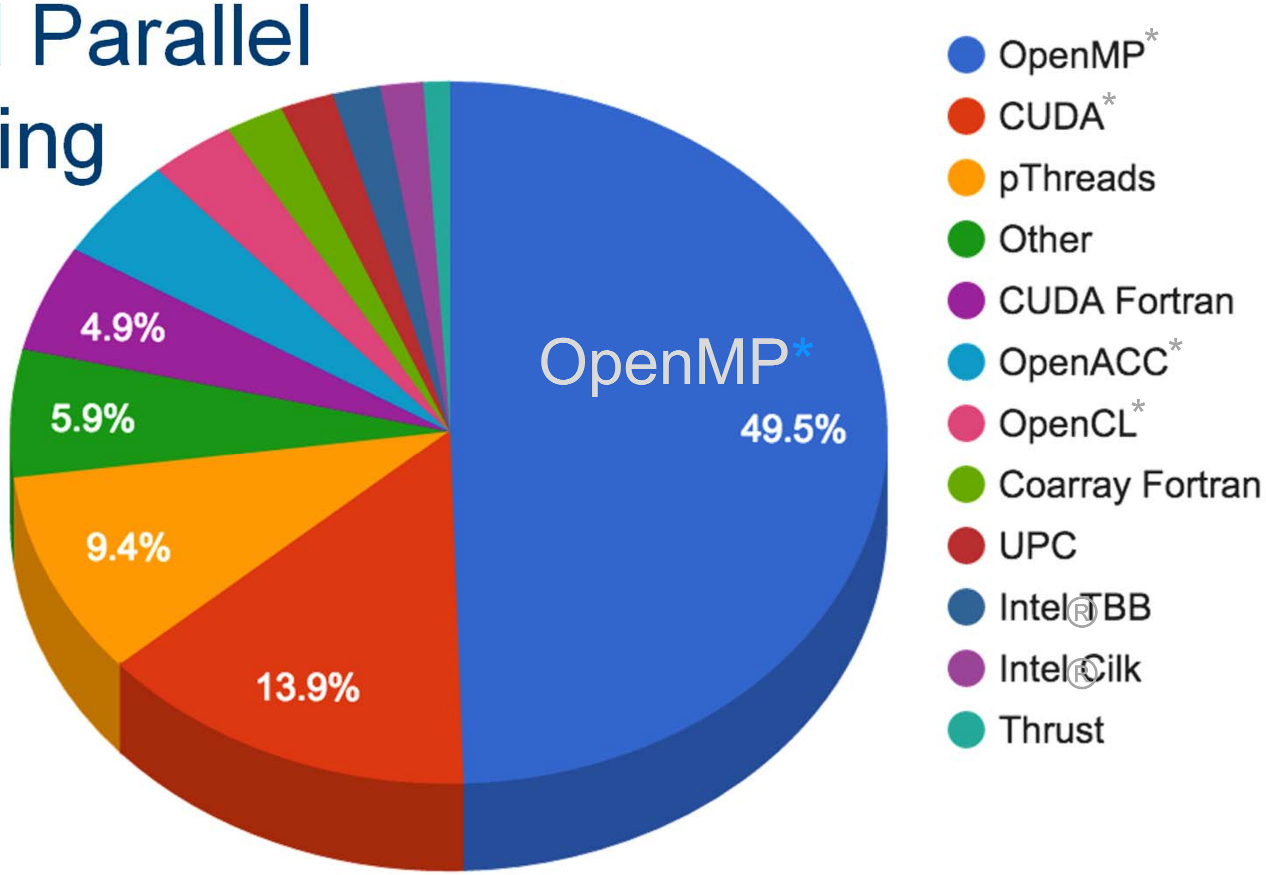
Data courtesy of Richard Gerber, National Energy Research Scientific Computing Center (NERSC), 2015.



Parallel Programming Models

Data courtesy of Richard Gerber, National Energy Research Scientific Computing Center (NERSC), 2015.

Node-level Parallel Programming Models



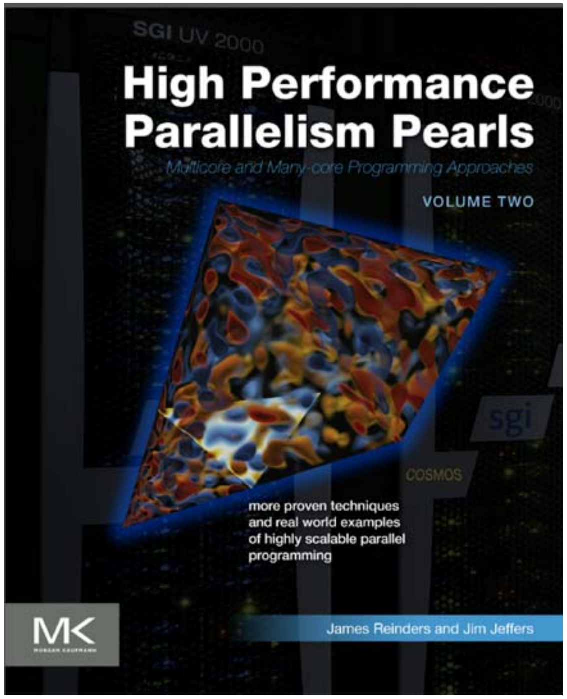
Data courtesy of Richard Gerber, National Energy Research Scientific Computing Center (NERSC), 2015.

Pearls Volume Two

25 chapters,
73 expert contributors.

Numerous “Real World” Code “Recipes” and examples using OpenMP*, MPI, OpenCL*, C, C++, Fortran.

Successful techniques, tips for key issues of scaling, locality of reference and vectorization


SGI UV 2000
High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
VOLUME TWO
sgi
COSMOS
more proven techniques and real world examples of highly scalable parallel programming
MK
MORGAN KAUFMANN
James Reinders and Jim Jeffers

“Modern processors are many-core, and will become moreso, that trend is simply not going away. Users and developers are flocking to these platforms. What we need to do to use them is not revolutionary, simply to make good use of techniques that have been known for a long time. But those techniques need to be used much more widely, with different emphasis, and at different scales.”

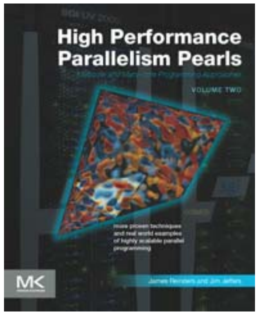
— Dan Stanzione
Executive Director
(Texas Advanced Computing Center
The University of Texas at Austin)

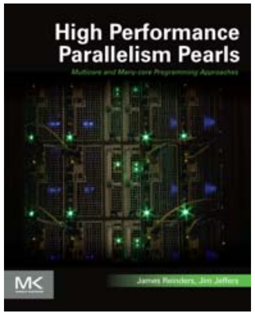
High Performance Programming Pearls Volume 2
Editors Jim Jeffers, James Reinders, (c) 2015, publisher: Morgan Kaufmann
Book cover design used with permission of publisher.

www.lotsofcores.com

All figures, diagrams and code freely downloadable.

25 Chapters – 24 discuss OpenMP


High Performance Parallelism Pearls
VOLUME TWO
MK
James Reinders and Jim Jeffers


High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
MK
James Reinders, Jim Jeffers

We saw example after example get performance and performance portability with "*just* parallel programming"

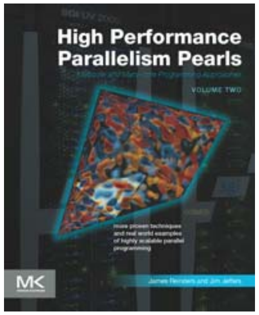
We summarize as "inspired by 61 cores"

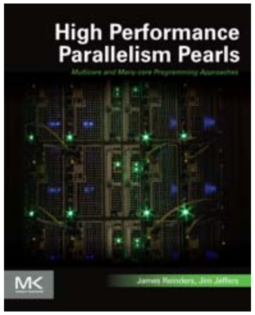
http:// lotsofcores.com

Volume 2 includes the following chapters:
Foreword by Dan Stanzione, TACC
Chapter 1: Introduction
Chapter 2: Numerical Weather Prediction Optimization
Chapter 3: WRF Goddard Microphysics Scheme Optimization
Chapter 4: Pairwise DNA Sequence Alignment Optimization
Chapter 5: Accelerated Structural Bioinformatics for Drug Discovery
Chapter 6: Amber PME Molecular Dynamics Optimization
Chapter 7: Low Latency Solutions for Financial Services
Chapter 8: Parallel Numerical Methods in Finance
Chapter 9: Wilson Dslash Kernel From Lattice QCD Optimization
Chapter 10: Cosmic Microwave Background Analysis: Nested Parallelism In Practice
Chapter 11: Visual Search Optimization
Chapter 12: Radio Frequency Ray Tracing
Chapter 13: Exploring Use of the Reserved Core
Chapter 14: High Performance Python Offloading
Chapter 15: Fast Matrix Computations on Asynchronous Streams
Chapter 16: MPI-3 Shared Memory Programming Introduction
Chapter 17: Coarse-Grain OpenMP for Scalable Hybrid Parallelism
Chapter 18: Exploiting Multilevel Parallelism with OpenMP
Chapter 19: OpenCL: There and Back Again
Chapter 20: OpenMP vs. OpenCL: Difference in Performance?
Chapter 21: Prefetch Tuning Optimizations
Chapter 22: SIMD functions via OpenMP
Chapter 23: Vectorization Advice
Chapter 24: Portable Explicit Vectorization Intrinsics
Chapter 25: Power Analysis for Applications and Data Centers

Book images used with permission.

27 Chapters – 24 discuss OpenMP




High Performance
Parallelism Pearls
Multicore and Many-core Programming Approaches
MK
James Reinders, Jim Jeffers

We saw example after example get performance and performance portability with "*just* parallel programming"

We summarize as "inspired by 61 cores"

http:// lotsofcores.com

Volume 1 includes the following chapters:
Foreword by Sverre Jarp, CERN.
Chapter 1: Introduction
Chapter 2: From 'Correct' to 'Correct & Efficient': A Hydro2D Case Study with Godunov's Scheme
Chapter 3: Better Concurrency and SIMD on HBM
Chapter 4: Optimizing for Reacting Navier-Stokes Equations
Chapter 5: Plesiochronous Phasing Barriers
Chapter 6: Parallel Evaluation of Fault Tree Expressions
Chapter 7: Deep-Learning and Numerical Optimization
Chapter 8: Optimizing Gather/Scatter Patterns
Chapter 9: A Many-Core Implementation of the Direct N-body Problem
Chapter 10: N-body Methods
Chapter 11: Dynamic Load Balancing Using OpenMP 4.0
Chapter 12: Concurrent Kernel Offloading
Chapter 13: Heterogeneous Computing with MPI
Chapter 14: Power Analysis on the Intel® Xeon Phi™ Coprocessor
Chapter 15: Integrating Intel Xeon Phi Coprocessors into a Cluster Environment
Chapter 16: Supporting Cluster File Systems on Intel® Xeon Phi™ Coprocessors
Chapter 17: NWChem: Quantum Chemistry Simulations at Scale
Chapter 18: Efficient Nested Parallelism on Large-Scale Systems
Chapter 19: Performance Optimization of Black-Scholes Pricing
Chapter 20: Data Transfer Using the Intel COI Library
Chapter 21: High-Performance Ray Tracing
Chapter 22: Portable Performance with OpenCL
Chapter 23: Characterization and Optimization Methodology Applied to Stencil Computations
Chapter 24: Profiling-Guided Optimization
Chapter 25: Heterogeneous MPI optimization with ITAC
Chapter 26: Scalable Out-of-Core Solvers on a Cluster
Chapter 27: Sparse Matrix-Vector Multiplication: Parallelization and Vectorization
Chapter 28: Morton Order Improves Performance

Book images used with permission.

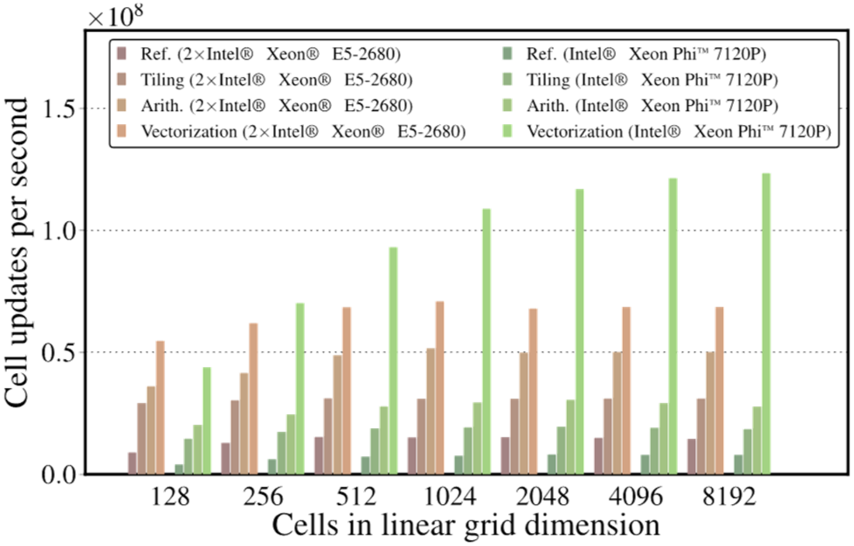
From ‘Correct’ to ‘Correct & Efficient’:

A Hydro2D case study with Godunov’s scheme
Volume 1, Chapter 2

Guillaume Colin de Verdière and Jason D. Sewall

High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
Chapter 2
MK
James Reinders, Jim Jeffers

- Real-world code developed in CEA
- Poetically notes, “*a rising tide lifts all boats*”, using:
  “*A common set of optimizations [that] benefit both general-purpose Intel® Xeon® processors and more specialized Intel® Xeon Phi™ accelerators*”
- **12x** increase on Intel Xeon Phi coprocessors
- **5x** increase on Intel Xeon processors



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. Configurations information is available in the book where the numbers are published.

Nested Parallelism

Nested parallelism is important to exploit.

Trending: more and more

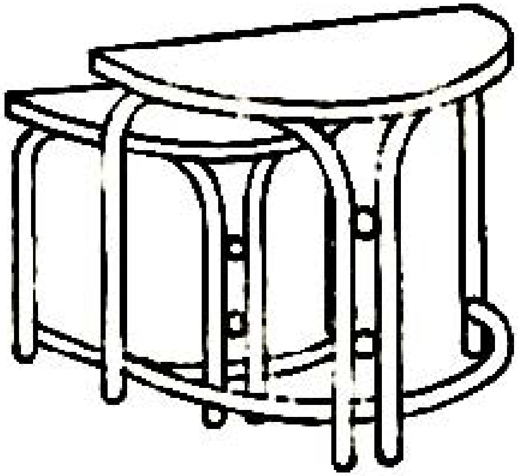
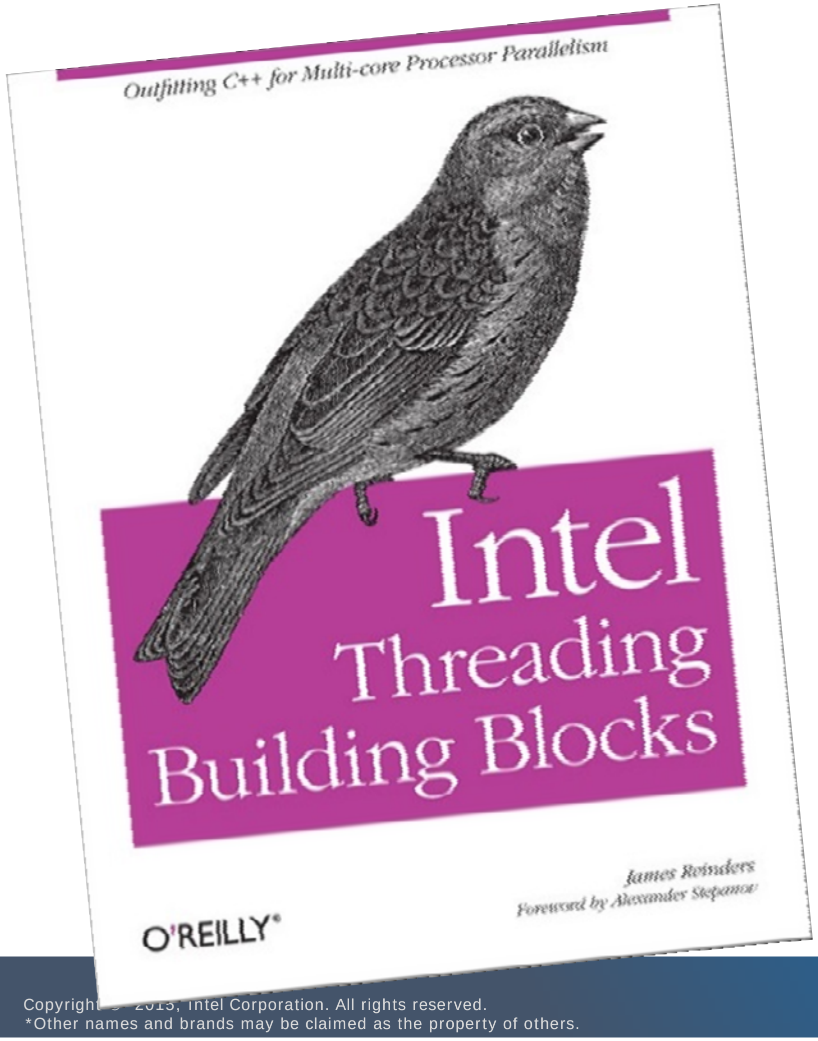


Image courtesy Smithsonian Museum
Cooper Hewitt, Smithsonian Design Museum, Accession Number 1998-71-28



Outfitting C++ for Multi-core Processor Parallelism

Intel Threading Building Blocks

James Reinders

Foreword by Alexander Stepanov

O'REILLY®

The MOST popular abstract parallelism model for C++

Nested Parallelism handle well.

OpenMP*

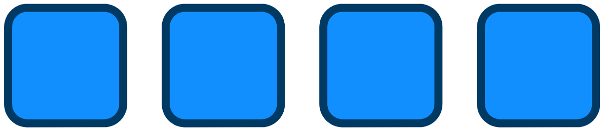
Not designed with nesting in mind.

All implementations turn off nested parallelism by default.

We NEED nested parallelism more and more.

So – you can turn it on a “little” ☺

OpenMP* Nested Parallelism: HOT TEAMS



OpenMP worker threads –
created ONCE PER PROGRAM

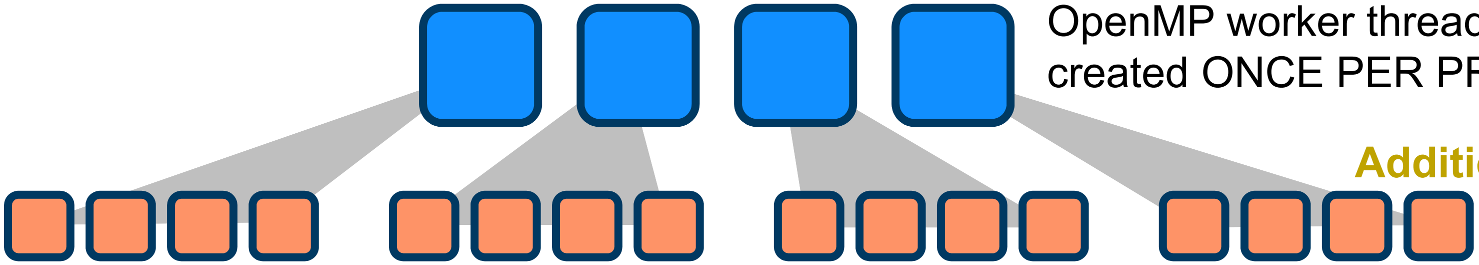
NESTED PARALLEL:

By DEFAULT, any parallel worker that executes a parallel construct does that work inside the same worker thread.

PRO: controlled memory footprint (including stack space)

CON: no load balancing

OpenMP* Nested Parallelism: HOT TEAMS



OpenMP worker threads – created ONCE PER PROGRAM

Additional level(s) created and released repeatedly

NESTED PARALLEL:

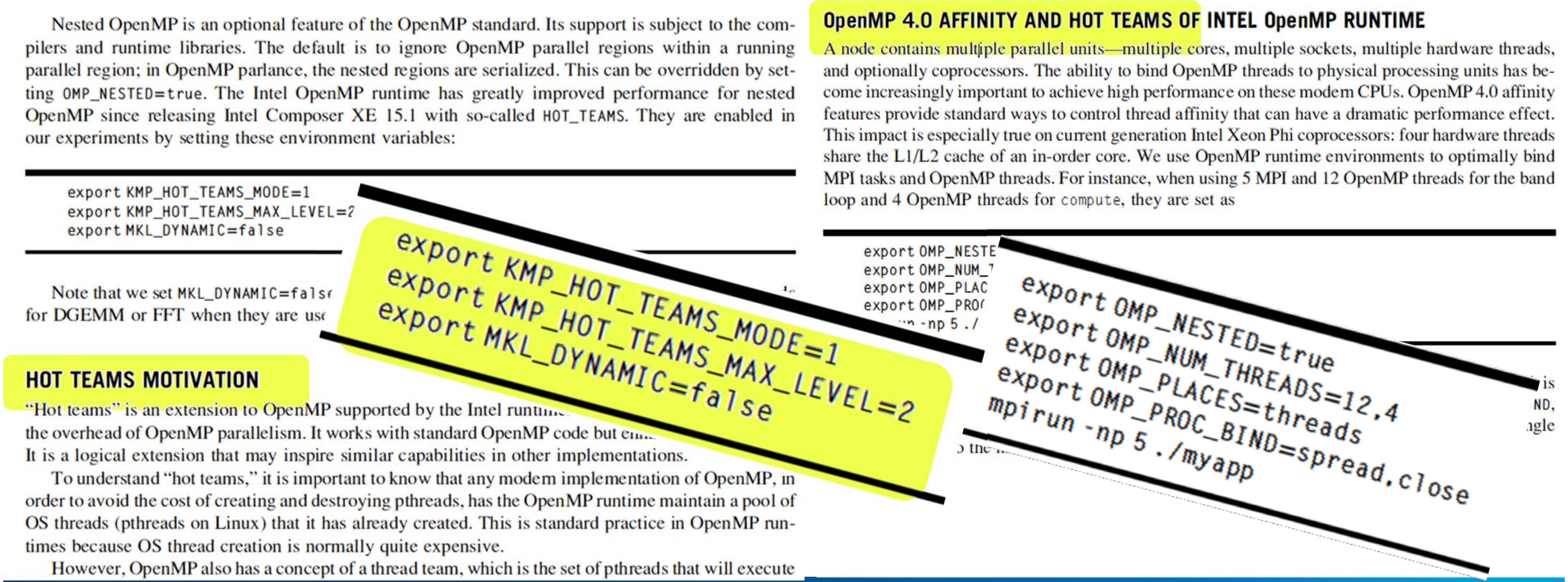
TURN ON NESTING (no code changes – done with environment variables)

PRO: load balancing

CON: high overhead, potential oversubscription (runaway memory/stack usage being the key issue)

OpenMP* Nested Parallelism: HOT TEAMS

Chapter 18: Exploiting Multilevel Parallelism with OpenMP*
Volume 2, Chapter 18



Nested OpenMP is an optional feature of the OpenMP standard. Its support is subject to the compilers and runtime libraries. The default is to ignore OpenMP parallel regions within a running parallel region; in OpenMP parlance, the nested regions are serialized. This can be overridden by setting OMP_NESTED=true. The Intel OpenMP runtime has greatly improved performance for nested OpenMP since releasing Intel Composer XE 15.1 with so-called HOT_TEAMS. They are enabled in our experiments by setting these environment variables:

```
export KMP_HOT_TEAMS_MODE=1
export KMP_HOT_TEAMS_MAX_LEVEL=2
export MKL_DYNAMIC=false
```

Note that we set MKL_DYNAMIC=fals
for DGEMM or FFT when they are us

HOT TEAMS MOTIVATION

“Hot teams” is an extension to OpenMP supported by the Intel runtim
the overhead of OpenMP parallelism. It works with standard OpenMP code but en
It is a logical extension that may inspire similar capabilities in other implementations.

To understand “hot teams,” it is important to know that any modern implementation of OpenMP, in order to avoid the cost of creating and destroying pthreads, has the OpenMP runtime maintain a pool of OS threads (pthreads on Linux) that it has already created. This is standard practice in OpenMP runtimes because OS thread creation is normally quite expensive.

However, OpenMP also has a concept of a thread team, which is the set of pthreads that will execute

OpenMP 4.0 AFFINITY AND HOT TEAMS OF INTEL OpenMP RUNTIME

A node contains multiple parallel units—multiple cores, multiple sockets, multiple hardware threads, and optionally coprocessors. The ability to bind OpenMP threads to physical processing units has become increasingly important to achieve high performance on these modern CPUs. OpenMP 4.0 affinity features provide standard ways to control thread affinity that can have a dramatic performance effect. This impact is especially true on current generation Intel Xeon Phi coprocessors: four hardware threads share the L1/L2 cache of an in-order core. We use OpenMP runtime environments to optimally bind MPI tasks and OpenMP threads. For instance, when using 5 MPI and 12 OpenMP threads for the band loop and 4 OpenMP threads for compute, they are set as

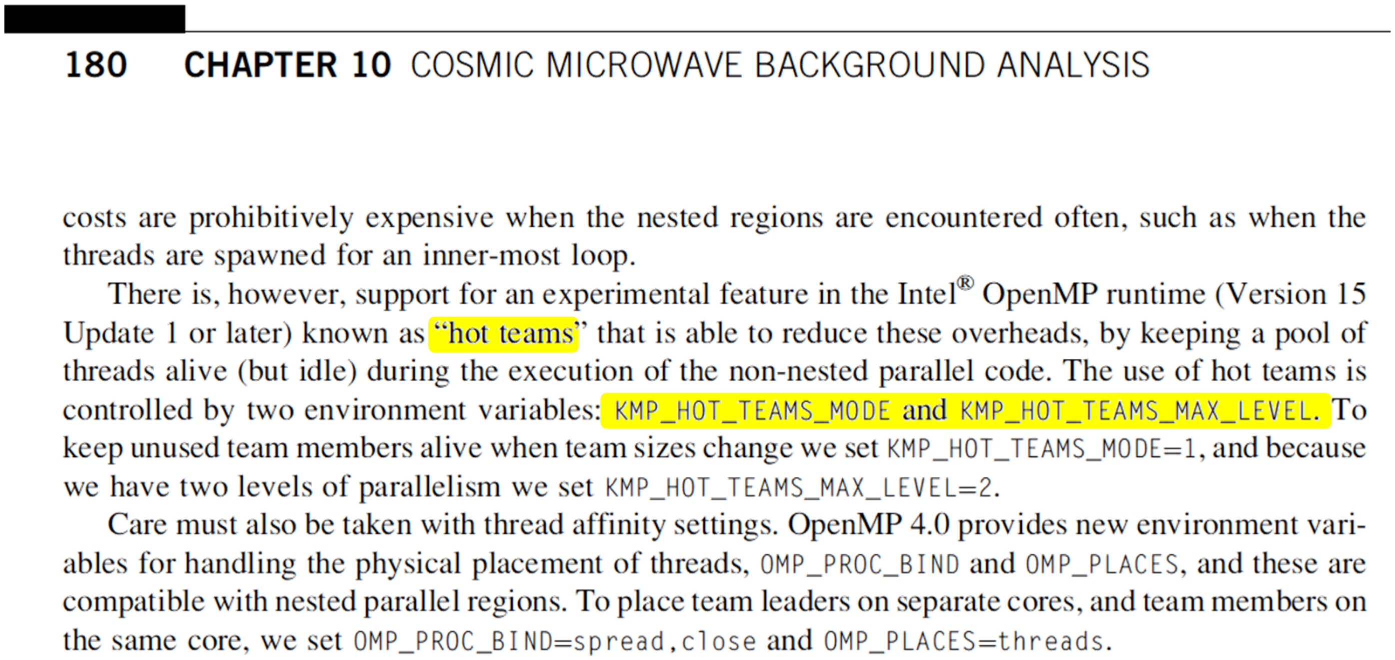
```
export OMP_NESTE
export OMP_NUM_T
export OMP_PLAC
export OMP_PROC
```

```
export KMP_HOT_TEAMS_MODE=1
export KMP_HOT_TEAMS_MAX_LEVEL=2
export MKL_DYNAMIC=false
```

```
export OMP_NESTED=true
export OMP_NUM_THREADS=12,4
export OMP_PLACES=threads
export OMP_PROC_BIND=spread,close
mpirun -np 5 ./myapp
```

OpenMP* Nested Parallelism: HOT TEAMS

Chapter 10: Cosmic Microwave Background Analysis: Nested Parallelism In Practice
Volume 2, Chapter 10

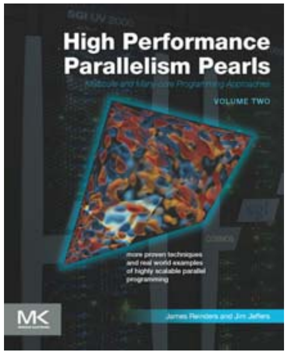


180 **CHAPTER 10** COSMIC MICROWAVE BACKGROUND ANALYSIS

costs are prohibitively expensive when the nested regions are encountered often, such as when the threads are spawned for an inner-most loop.

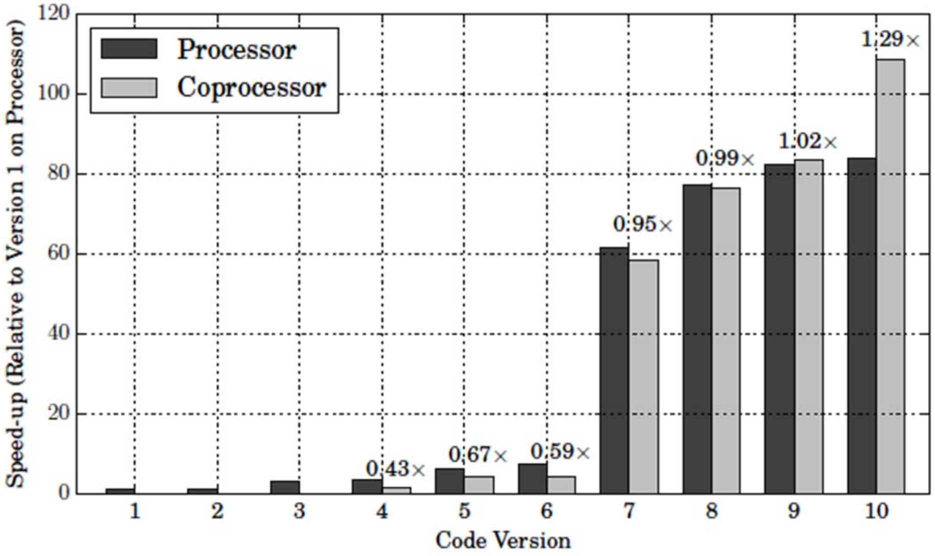
There is, however, support for an experimental feature in the Intel® OpenMP runtime (Version 15 Update 1 or later) known as "hot teams" that is able to reduce these overheads, by keeping a pool of threads alive (but idle) during the execution of the non-nested parallel code. The use of hot teams is controlled by two environment variables: KMP_HOT_TEAMS_MODE and KMP_HOT_TEAMS_MAX_LEVEL. To keep unused team members alive when team sizes change we set KMP_HOT_TEAMS_MODE=1, and because we have two levels of parallelism we set KMP_HOT_TEAMS_MAX_LEVEL=2.

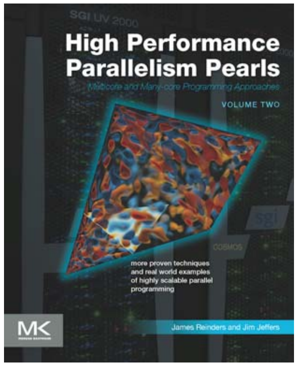
Care must also be taken with thread affinity settings. OpenMP 4.0 provides new environment variables for handling the physical placement of threads, OMP_PROC_BIND and OMP_PLACES, and these are compatible with nested parallel regions. To place team leaders on separate cores, and team members on the same core, we set OMP_PROC_BIND=spread,close and OMP_PLACES=threads.


High Performance Parallelism Pearls
VOLUME TWO
MK
James Reinders and Jim Jeffers

High Performance Parallelism Pearls

Cosmic Microwave Background Analysis: Nested Parallelism in Practice Volume 2, Chapter 10

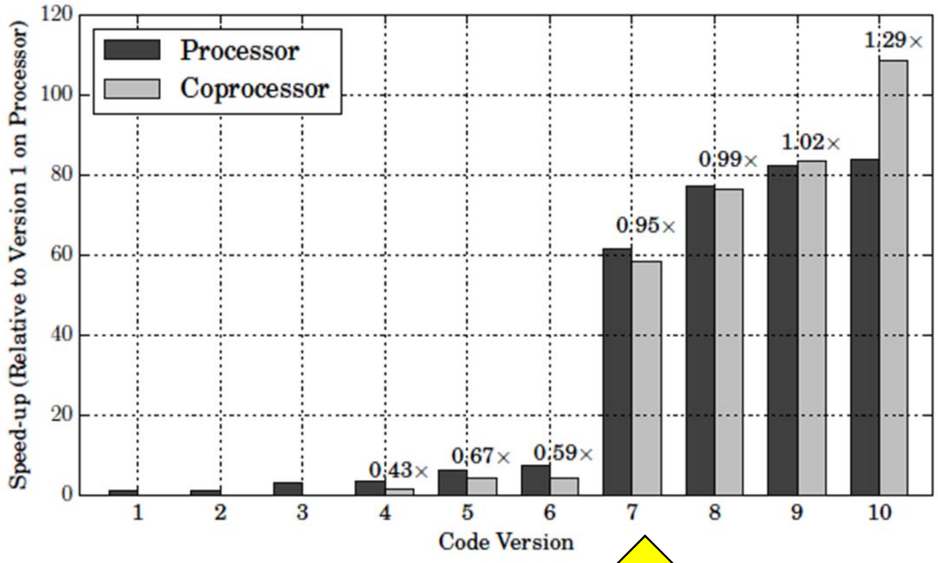



SGI UV 2000
High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
VOLUME TWO
more proven techniques and real world examples of highly scalable parallel programming
MK
James Reinders and Jim Jeffers

High Performance Parallelism Pearls

Cosmic Microwave Background Analysis: Nested Parallelism in Practice

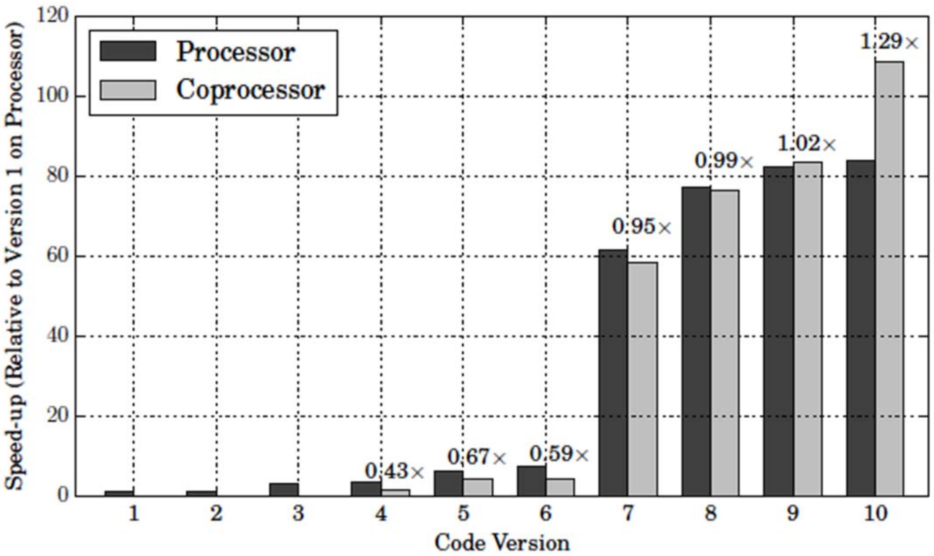
Volume 2, Chapter 10



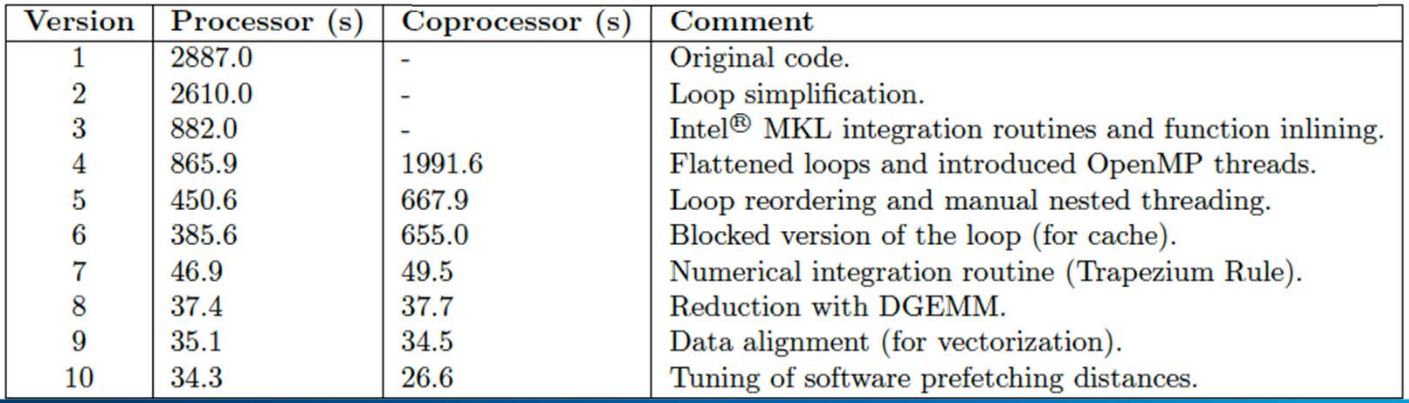
Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors 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. Configurations information is available in the book where the numbers are published.

High Performance Parallelism

Cosmic Microwave Background Nested Parallelism in Practice Volume 2, Chapter 10



We find that using a simple trapezium rule integrator combined with hand-selected sampling points (to improve accuracy in areas of interest) provides sufficient numerical accuracy to obtain a physically meaningful result, and the reduced space and time requirements of this simplified method give a speed-up of O(10x).



Version	Processor (s)	Coprocessor (s)	Comment
1	2887.0	-	Original code.
2	2610.0	-	Loop simplification.
3	882.0	-	Intel® MKL integration routines and function inlining.
4	865.9	1991.6	Flattened loops and introduced OpenMP threads.
5	450.6	667.9	Loop reordering and manual nested threading.
6	385.6	655.0	Blocked version of the loop (for cache).
7	46.9	49.5	Numerical integration routine (Trapezium Rule).
8	37.4	37.7	Reduction with DGEMM.
9	35.1	34.5	Data alignment (for vectorization).
10	34.3	26.6	Tuning of software prefetching distances.

OpenMP* vs. OpenCL™

OpenCL™: There and Back Again
Volume 2, Chapter 19

Matthias Noack, Florian Wende, Klaus-Dieter Oertel

Optimize Hexciton kernel of the GPU-HEOM code. Optimization of code for coprocessor yields 5.9x for OpenCL™, 6.4x for OpenMP*.

“Obviously, there is no such thing as portable performance for the Hexciton kernel.”

The “CPU” kernel:

- Performs well on Intel® Xeon® processor and Intel® Xeon Phi™ coprocessor. The GPGPU does not do well with [it]

The “GPU” kernel:

- Surprisingly good device-utilization on the Intel Xeon processor.
- Not on the coprocessor

Op in

Matth ertel

Optimize Hexciton ker of code for coprocessor yields 5.9

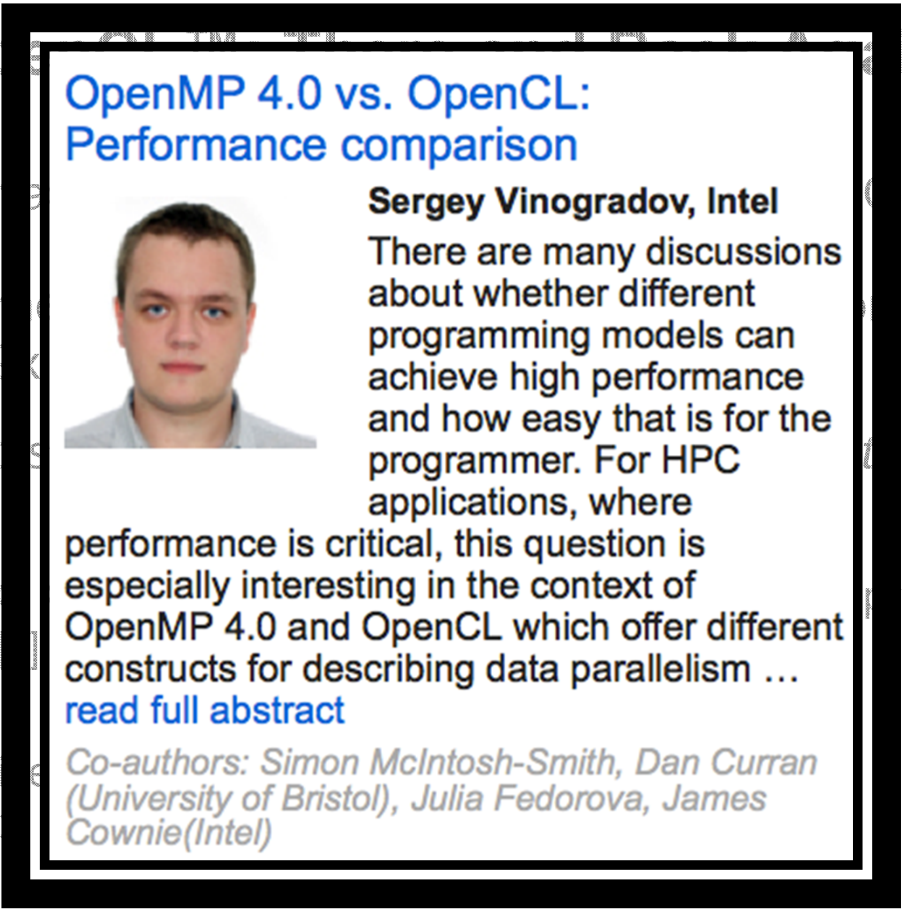
“Obviously, there is no e Hexciton kernel.”

The “CPU” kernel:

- Performs well on Intel ocessor. The GPGPU does not do well with

The “GPU” kernel:

- Surprisingly good dev
- Not on the coprocesso



OpenMP 4.0 vs. OpenCL: Performance comparison

Sergey Vinogradov, Intel

There are many discussions about whether different programming models can achieve high performance and how easy that is for the programmer. For HPC applications, where performance is critical, this question is especially interesting in the context of OpenMP 4.0 and OpenCL which offer different constructs for describing data parallelism …

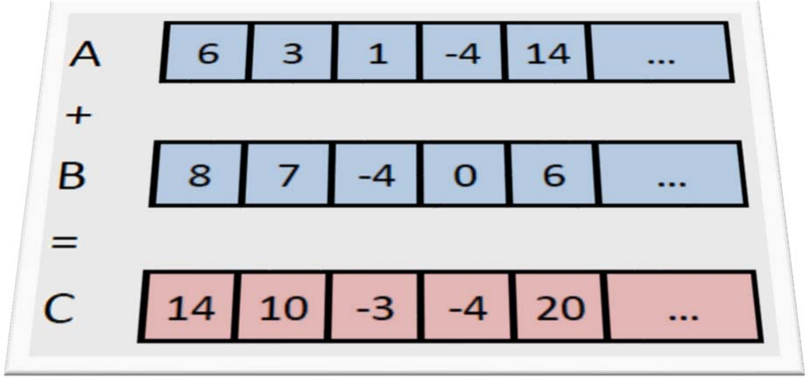
read full abstract

Co-authors: Simon McIntosh-Smith, Dan Curran (University of Bristol), Julia Fedorova, James Cownie(Intel)

Vectorization

Nested parallelism is important to exploit.

Trending: more and more so.

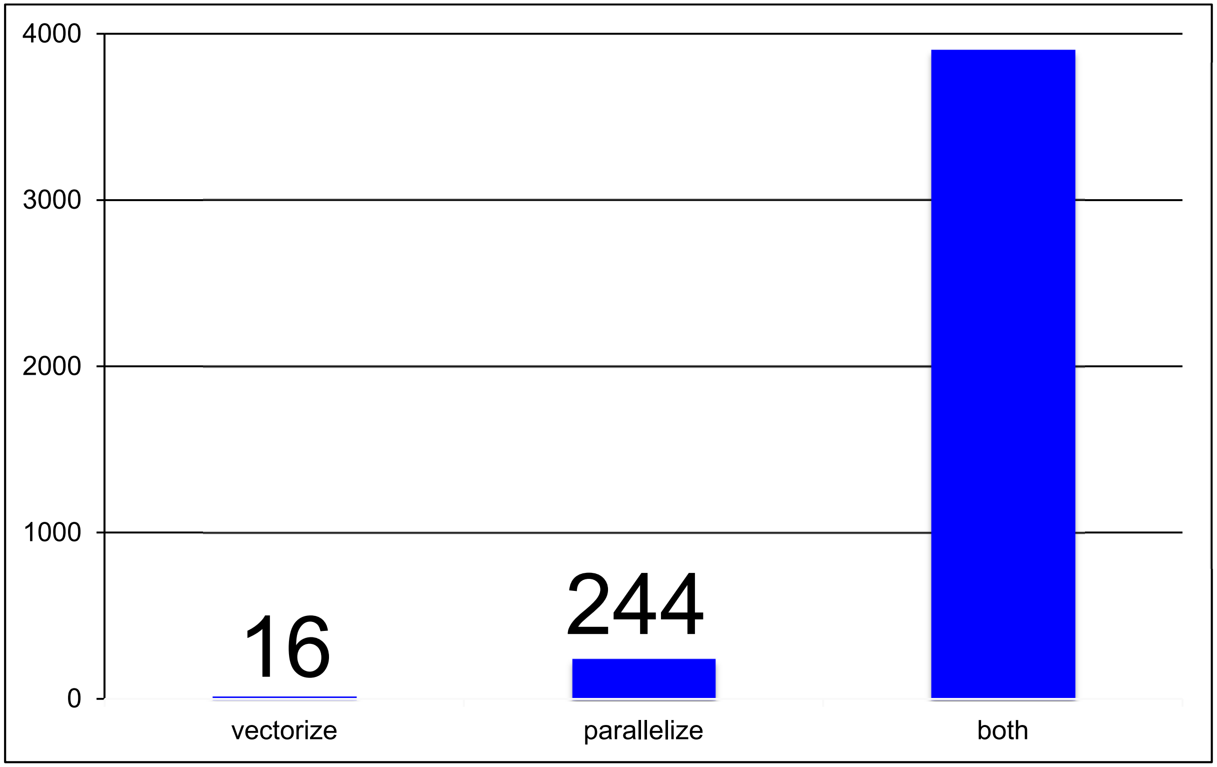


A	6	3	1	-4	14	...
+						
B	8	7	-4	0	6	...
=						
C	14	10	-3	-4	20	...

Volume Two…

- Chapters 2-4, 8
- SIMD functions 22
- Vectorization Advisor 23
- OpenVec 24

16 x 244 = 3904



Multithreading

Vectorization

Assertion:

We need to embrace *explicit* vectorization in our programming.

- Abstraction levels vary

OpenMP* 4.0: #pragma omp simd

```
void v_add (float *c,
        float *a,
        float *b)
{
#pragma omp simd
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

OpenMP* 4.0: #pragma omp simd

```
void v_add(
        fl
        fl
{
#pragma om
    for (i
        c[i
}
```

SIMD Programming with OpenMP

Georg Zitzelsberger, Intel Corp.

If you think OpenMP is merely about threading then you might be interested in the latest features of OpenMP 4.x that exploit the SIMD capabilities of modern processors. Since processors tend to spend more die space for SIMD, growing with every new generation, the so-called “vectorization” becomes more important … read full abstract

Accelerated Structural Bioinformatics for Drug Discovery

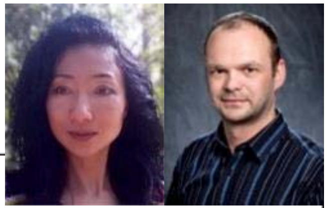
Volume 2, Chapter 5

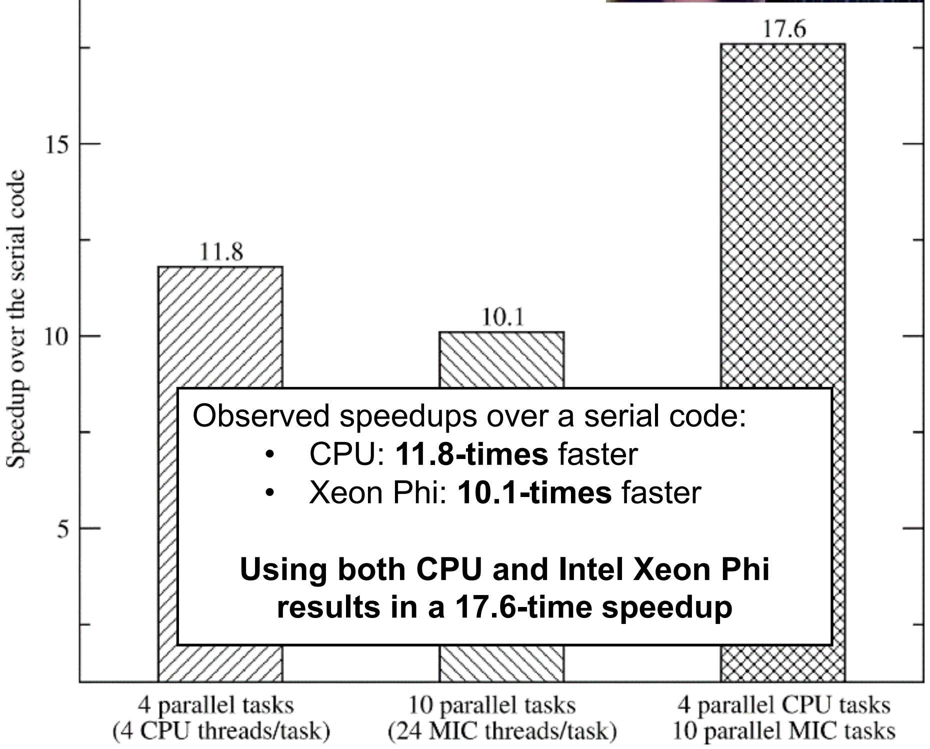
Wei P. Feinstein, Michal Brylinski

- Describes the development and benchmarking of eFindSite, a structural bioinformatics algorithm
  - eFindSite identifies drug-binding sites in proteins and molecular fingerprint-based virtual screening

- Demonstrates the use of thread-private copies to eliminate use of thread-unsafe common blocks

- As a warning to readers, the authors note:
  1. Use common blocks and not parameter passing
     - "Spaghetti-like" code makes parameter passing difficult and time consuming
  2. Use of OMP_STACKSIZE may not be sufficient to increase the stack size
     - May need to use "ulimit –s" as well on Linux





Observed speedups over a serial code:

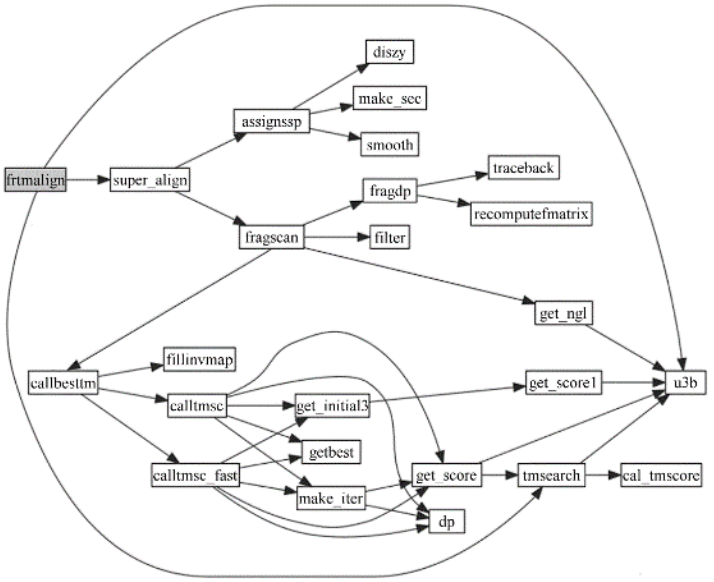
- CPU: **11.8-times** faster
- Xeon Phi: **10.1-times** faster

Using both CPU and Intel Xeon Phi results in a 17.6-time speedup

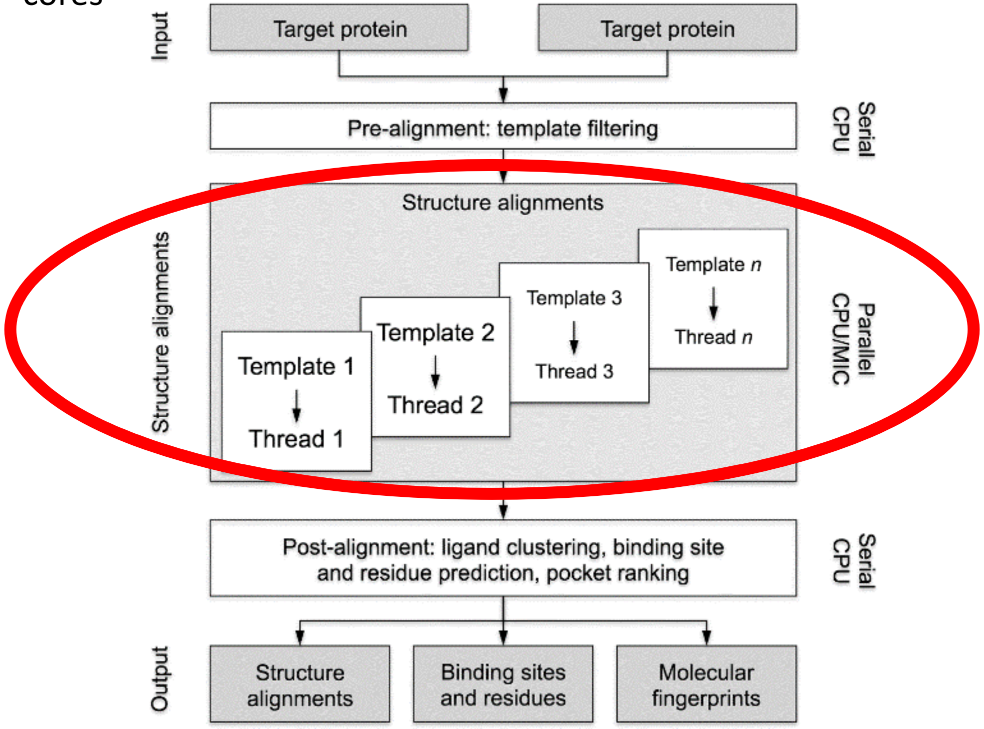
Finding the hotspots

The framework of eFindSite is written in C++, whereas protein structure alignments are implemented in Fortran77

- Profiling reveals that 90% of the runtime is consumed by structure alignment calculations
- Parallelizing these calculations using OpenMP pragmas enables performance improvements that scale well with the number of cores



Call graph of subroutines for protein structure alignments



Workflow of eFindSite

Coarse-Grain OpenMP* for Scalable Hybrid Parallelism

Volume 2, Chapter 17

Enda O'Brien

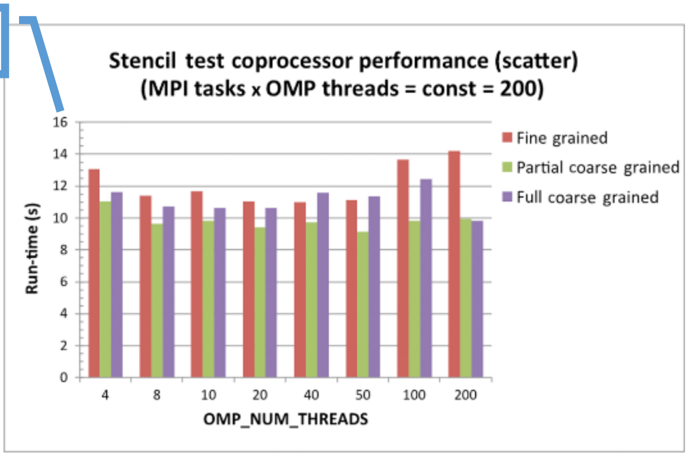
Used a very simple FORTRAN program that performs repeated smoothing or "filtering" of an arbitrarily large three-dimensional array

- Representative of "stencil" operations widely used in many different applications
- Example code is parallelized with MPI, fine-grained OpenMP , and coarse-grained OpenMP

Discusses how the performance of simple loop-level or "fine-grained" OpenMP directives can be significantly improved by merging and expanding the OpenMP parallel regions into a single large "coarse-grained" parallel region

- MPI latency is much larger within a coprocessor than within the host node (approx. 2.5 vs. 0.5µs)
- Results suggest that a balanced "mix" of OpenMP threads and MPI processes is usually better than putting most or all the parallelism into either OpenMP or MPI

16s



Stencil-test run-times on Intel Xeon Phi coprocessor using 200 threads

OpenMP* Future

OpenMP* Future

What about 100 years from now?

2115?

100 years from now: 2115

19 years ago: 1996 – OpenMP concept

20?

35?

100 years from now: 2115

19 years ago: 1996 – OpenMP concept

20?

35?

38 years ago: 1977 – Fortran 77 namesake

100 years from now: 2115

19 years ago: 1996 – OpenMP concept

20?

35?

38 years ago: 1977 – Fortran 77 namesake

50?

59 years ago: 1956 – Fortran specification

My “wish” list for OpenMP

Everywhere

Forever

My “wish” list for OpenMP

Everywhere

- CPUs, GPUs, FPGAs…

Forever

My “wish” list for OpenMP

Everywhere

- CPUs, GPUs, FPGAs…

Forever

- Composability

My “wish” list for OpenMP

Everywhere

- CPUs, GPUs, FPGAs…

Forever

- Composability
- Respect legacy, but retire it (within 100 years)

Beginning OpenMP

C$OMP PARALLEL FOR...

...

C$OMP PARALLEL FOR...

...

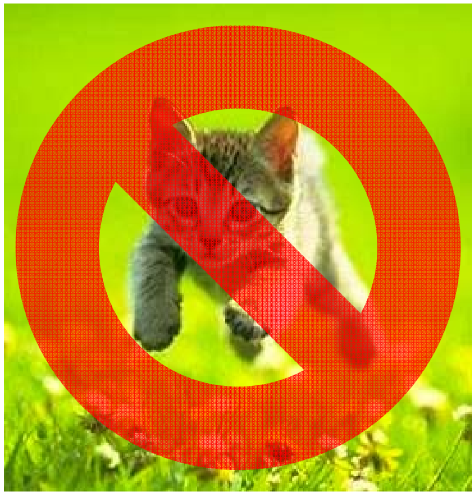
C$OMP PARALLEL FOR...

...

C$OMP PARALLEL FOR...

...

Advanced OpenMP



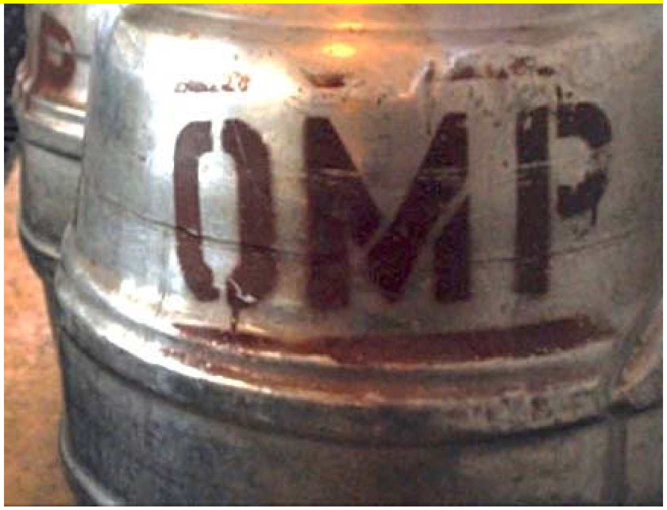
Teach OpenMP on

- CPUs, GPUs, and FPGAs
- Move away from barrier based algorithms
  - A candidate: to a flow graph based algorithm
- Composability

OLD MARKET PUB
BREWERY • GRILL • FAMILY DINING
OLD MARKET PUB
OMP
& BREWERY

My “wish” list for OpenMP

Q&A time



Everywhere

- CPUs, GPUs, FPGAs…

Forever

- Composability
- Respect legacy, but retire it (within 100 years)

james.r.reinders@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.

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

intel®