Computer Architecture and Structured Parallel Programming
James Reinders, Intel
Parallel Computing
CIS 410/510
Department of Computer and Information Science
Computer Architecture and Structured Parallel Programming

James Reinders, Intel
Computer Architecture & Structured Parallel Programming

- review aspects of computer architecture that are critical to high performance computing
- discuss how to think about best algorithm design using structured parallel programming techniques
- task vs. data parallelism and why data parallelism is key
- introduce TBB, OpenMP*
- introduce Intel® Xeon Phi™ architecture.
Computer Architecture and Structured Parallel Programming

James Reinders, Intel
See the Forest
A cliché about someone missing the “big picture” because they focus too much on details:

They “cannot see the forest for the trees.”
See the Forest

I ❤️ architecture.
I ❤️ architecture.
but...
Can you teach parallel programming without first teaching computer architecture?
Can you teach parallel programming without first teaching computer architecture? (Or without just teaching a single API?)
See the Forest

- TREES
- Cores
- HW threads
- Vectors
- Offload
- Heterogeneous
- Cloud
- Caches
- NUMA
<table>
<thead>
<tr>
<th>TREES</th>
<th>FOREST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>HW threads</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Vectors</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Offload</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Cloud</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Caches</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>NUMA</td>
<td>Parallelism, Locality</td>
</tr>
</tbody>
</table>
See the Forest

**Advice:** proper abstractions
- Use tasks
- Use SIMD (10:30 talk)
- Avoid, Use TARGET
- Avoid via neo-hetero
- What's a cloud?
- Use abstractions

**TREES**
- Cores
- HW threads
- Vectors
- Offload
- Heterogeneous
- Cloud
- Caches
- NUMA
<table>
<thead>
<tr>
<th>TREES</th>
<th>FOREST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>HW threads</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Vectors</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Offload</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Cloud</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>Caches</td>
<td>Parallelism, Locality</td>
</tr>
<tr>
<td>NUMA</td>
<td>Parallelism, Locality</td>
</tr>
</tbody>
</table>

See the Forest

© 2014, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Teach the Forest

Increase exposing parallelism.
Increase locality of reference.
Teach the Forest

Increase exposing parallelism.
Increase locality of reference.

Why? Because it’s programming that addresses the universal needs of computers today and in the future future.
Teach the Forest

Increase exposing parallelism.
Increase locality of reference.

THIS IS YOUR MISSION
Why so many cores?
Why Multicore?

The “Free Lunch” is over, really.

But Moore’s Law continues!
Processor Clock Rate over Time

Growth halted around 2005
Transistors per Processor over Time
Continues to grow exponentially (Moore’s Law)
Moore’s Law

Number of components (transistors) doubles about every 18-24 months.
Parallelism is key +
Exploit locality of data
Parallelism is key
Is this the Architecture Track?
These were simpler times.
CPU + cache

Memories got “further away” (meaning: CPU speed increased faster than memory speeds)

A closer “cache” for frequently used data helps performance when memory is no longer a single clock cycle away.
CPU + caches

Memories keep getting “further away” (this trend continues today).

More “caches” help even more (with temporal reuse of data).
CPU with caches

As transistor density increased (Moore’s Law), cache capabilities were integrated onto CPUs. Higher performance external (discrete) caches persisted for some time while integrated cache capabilities increase.
CPU / Coprocessors

Coprocessors appearing first in 1970s were FP accelerators for CPUs without FP capabilities.
CPU / Coprocessors

As transistor density increased (Moore’s Law), FP capabilities were integrated onto CPUs. Higher performance discrete FP “accelerators” persisted a little bit while integrated FP capabilities increase.
CPU / Coprocessors

Interest to provide hardware support for displays increased as use of graphics grew (games being a key driver).

This led to graphics processing units (GPUs) attached to CPUs to create video displays.
CPU / Coprocessors

GPU speeds and CPU speeds increase faster than memory speeds. Direct connection to memory best done via caches (on the CPU).
CPU / Coprocessors

GPU speeds and CPU speeds increase faster than memory speeds. Direct connection to memory best done via caches (on the CPU).
CPU / Coprocessors

As transistor density increased (Moore's Law), GPU capabilities were integrated onto CPUs. Higher performance external (discrete) GPUs persist while integrated GPU capabilities increase.
CPU / Coprocessors

A many core coprocessor (Intel® Xeon Phi™) appears, purpose built for accelerating technical computing.
CPU / Coprocessors

As transistor density increased (Moore’s Law), many core capabilities will be integrated to create a many core CPU. (“Knights Landing”)
Nodes

“Nodes” are building blocks for clusters. With or without GPUs. Displays not needed.
Clusters

Clusters are made by connecting nodes - regardless of “Nodes” type.
NIC (Network Interface Controller) integration

As transistor density increased (Moore’s Law), NIC capabilities will be integrated onto CPUs.
What matters when programming?

- Parallelism
- Locality
Amdahl who?
How much parallelism is there?

Amdahl’s Law

Gustafson’s observations on Amdahl’s Law
Work 500 Time 400
Speedup 1.25X
Work 500 Time 350  
Speedup 1.4X
many processing cores ~0 time

Work 500 Time 300
Speedup 1.7X
Amdahl’s law

“...the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.”

– Amdahl, 1967
Amdahl’s law – an observation

“...speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.”

– Gustafson, 1988
Work 500 Time 500
Speedup 1X
Work 700 Time 500
Speedup 1.4X
Work $2N^2 + 100 + 300$ Time $500$
Speedup $O(N)$
How much parallelism is there?

Amdahl’s Law

Gustafson’s observations on Amdahl’s Law

Plenty –

but the workloads need to continue to grow!
Why Intel® Xeon Phi™?
Intel® Xeon Phi™ Coprocessor

It’s just a different design point.
Not a different programming paradigm.

Little cores vs. big cores. All x86.

vs.
Performance

\[
\frac{\text{Work}}{\text{Time}} = \frac{\text{Work}}{\text{Instructions}} \times \frac{\text{Instruction}}{\text{Cycle}} \times \frac{\text{Cycle}}{\text{Time}}
\]

Better algorithm → same work with fewer instructions

The compiler can optimize for fewer instructions, choose instructions with better IPC

**Cache efficient algorithms**: higher IPC

**Vectorization**: same work with fewer instructions

**Parallelization**: more instructions per cycle
Remember Pollack’s rule: Performance ~

4x the die area gives 2x the performance in one core, but
4x the performance when dedicated to 4 cores

Conclusions (with respect to Pollack’s rule)
A powerful handle to adjust
“Performance/Watt”
Weaker cores can be beneficial
(but many of them)

→ Parallel hardware
→ Parallel algorithms
→ Appropriate tools
Speedup?

Peak perf. by example ([http://ark.intel.com/](http://ark.intel.com/))

- Intel Xeon E5-2680 (not the top-bin)
  2S x 8C x 2.7 GHz x 4FDP x 2 ops* → ~345 GF/s

- Intel Xeon Phi 3120A (lowest bin)
  57C x 1.1 GHz x 8FDP x 2 ops* → ~1 TF/s

**Amdahl's Law** determines the total speedup $S^*$ with $S^* = 1 / [(1-P) + P/S]$ of a mixture of serial and parallel code sections with the parallel speedup $S$ and an amount of parallel code $P$ (strong scaling).
Picture worth many words

© 2013, James Reinders & Jim Jeffers, diagram used with permission
Intel® Xeon Phi™ Coprocessors

Highly-parallel Processing for Unparalleled Discovery

**Groundbreaking: differences**

- Up to 61 IA cores/1.1 GHz/ 244 Threads
- Up to 8GB memory with up to 352 GB/s bandwidth
- 512-bit SIMD instructions
- Linux operating system, IP addressable
- Standard programming languages and tools

**Leading to Groundbreaking results**

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 1 TeraFlop/s double precision peak performance¹</td>
<td></td>
</tr>
<tr>
<td>Enjoy up to 2.2x higher memory bandwidth than on an Intel® Xeon® processor E5 family-based server.²</td>
<td></td>
</tr>
<tr>
<td>Up to 4x more performance per watt than with an Intel® Xeon® processor E5 family-based server.³</td>
<td></td>
</tr>
</tbody>
</table>

¹ 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, operating systems, and workloads. 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.
² For more information go to http://www.intel.com/products. Intel Xeon, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Knights Corner Micro-architecture
Knights Corner Core

- x86 specific logic < 2% of core + L2 area

- L2 Ctl
- L2 TLB
- 512KB L2 Cache
- To On-Die Interconnect

- PPF
- PF
- D0
- D1
- D2
- E
- WB
Vector Processing Unit

PPF  PF  D0  D1  D2  E  WB
D2  E  VC1  VC2  V1-V4  WB

D2  E  VC1  VC2  V1  V2  V3  V4

DEC  VPU  RF  3R, 1W
LD
EMU
ST
Mask RF
Scatter Gather

Vector ALUs
- 16 Wide x 32 bit
- 8 Wide x 64 bit
- Fused Multiply Add

Copyright © 2012 Intel Corporation. All rights reserved.
Distributed Tag Directories

Tag Directories track cache-lines in all L2s
Interconnect: 2X AD/AK
Caches - For or Against?

Caches:
- high data BW
- low energy per byte of data supplied
- programmer friendly (coherence just works)

Coherent Caches are a key MIC Architecture Advantage

Results have been simulated and are provided for informational purposes only. Results were derived using simulations run on an architecture simulator or model. Any difference in system hardware or software design or configuration may affect actual performance.
it is an SMP-on-a-chip running Linux
vision
span from few cores to many cores with consistent models, languages, tools, and techniques
Illustrative example

Fortran code using MPI, single threaded originally. Run on Intel® Xeon Phi™ coprocessor natively (no offload).

Based on an actual customer example. Shown to illustrate a point about common techniques. Your results may vary!
Illustrative example

Fortran code using MPI, single threaded originally. Run on Intel® Xeon Phi™ coprocessor natively (no offload).
Illustrative example

Fortran code using MPI, single threaded originally. Run on Intel® Xeon Phi™ coprocessor natively (no offload).

Common optimization techniques... “dual benefit”
Illustrative example

Fortran code using MPI, single threaded originally.
Run on Intel® Xeon Phi™ coprocessor natively (no offload).

Common optimization techniques...
“dual benefit”
Top 500 (June 2014): Again... the #1 system (third time) is a Neo-heterogeneous system (Common Programming Model) (Intel® Xeon® Processors + Intel® Xeon Phi™ Coprocessor)

Source: June 2014 “Top 500” - www.top500.org
Knights Landing
(Next Generation Intel® Xeon Phi™ Products)

Platform Memory: DDR4 Bandwidth and Capacity Comparable to Intel® Xeon® Processors

Compute:
- Energy-efficient IA cores
- Microarchitecture enhanced for HPC
- 3X Single Thread Performance vs Knights Corner
- Intel Xeon Processor Binary Compatible

On-Package Memory:
- Up to 16GB at launch
- 5X Bandwidth vs DDR4
- 1/3X the Space
- 5X Power Efficiency

Jointly Developed with Micron Technology

3+ TFLOPS
In One Package
Parallel Performance & Density

2nd half '15
1st commercial systems

Source: June 2014 Intel @ ISC’14
How do I “think parallel”?
Map

- *Map* invokes a function on every element of an index set.

- The index set may be abstract or associated with the elements of an array.

- Corresponds to “parallel loop” where iterations are independent.

**Examples:** gamma correction and thresholding in images; color space conversions; Monte Carlo sampling; ray tracing.
• *Reduce* combines every element in a collection into one using an *associative* operator:
  \[ x + (y + z) = (x + y) + z \]

• For example: *reduce* can be used to find the sum or maximum of an array.

• Vectorization may require that the operator *also* be *commutative*:
  \[ x + y = y + x \]

**Examples:** averaging of Monte Carlo samples; convergence testing; image comparison metrics; matrix operations.
Stencil

- *Stencil* applies a function to neighbourhoods of an array.
- Neighbourhoods are given by set of relative offsets.
- Boundary conditions need to be considered.

**Examples:** image filtering including convolution, median, anisotropic diffusion
Pipeline

- *Pipeline* uses a sequence of stages that transform a flow of data

- Some stages may retain state

- Data can be consumed and produced incrementally: “online”

**Examples:** image filtering, data compression and decompression, signal processing
Parallelize pipeline by
- Running different stages in parallel
- Running *multiple copies* of stateless stages in parallel

Running multiple copies of stateless stages in parallel requires reordering of outputs

Need to manage buffering between stages
Structured Parallel Programming
- Michael McCool
- Arch Robison
- James Reinders

Uses Cilk Plus and TBB as primary frameworks for examples.
Appendices concisely summarize Cilk Plus and TBB.
www.parallelbook.com
Use abstractions !!!
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th></th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP*</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Use abstractions !!!

Avoid direct programming to the low level interfaces (like pthreads).

PROGRAM IN TASKS, NOT THREADS

Is OpenCL* low level? For HPC – YES.
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Choose First (limited functions)
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Choose First (limited functions)  
Cluster (distributed memory)
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Choose First (limited functions)

Cluster (distributed memory)

Node (shared memory)
We asked ourselves:

- How should C++ be extended?
  - “templates / generic programming”
- What do we want to solve?
  - Abstraction with good performance (scalability)
  - Abstraction that steers toward easier (less) debugging
  - Abstraction that is readable
Generic Parallel Algorithms
Efficient scalable way to exploit the power of multi-core without having to start from scratch

Concurrent Containers
Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety

Flow Graph
A set of classes to express parallelism via a dependency graph or a data flow graph

Thread Local Storage
Supports infinite number of thread local data

Task Scheduler
Sophisticated engine with a variety of work scheduling techniques that empowers parallel algorithms & the flow graph

Synchronization Primitives
Atomic operations, several flavors of mutexes, condition variables

Memory Allocation
Per-thread scalable memory manager and false-sharing free allocators

Thread-safe timers
Threads
OS API wrappers
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Choose First (limited functions)

Cluster (distributed memory)

Node (shared memory)
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP*</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Choose First (limited functions)
Cluster (distributed memory)
Node (shared memory)

Up and coming for C++ (keywords, compilers)
Because... you just have to expect “more”
Affect future C++ standards? (2021?)
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>implemented</td>
<td>vendor libraries</td>
<td>many</td>
<td>in compiler</td>
<td>portable</td>
<td>in compiler</td>
</tr>
<tr>
<td>supported by</td>
<td>most vendors</td>
<td>open src &amp; vendors</td>
<td>most compilers</td>
<td>ported most everywhere</td>
<td>gcc and Intel (llvm future)</td>
</tr>
</tbody>
</table>

Compare...

<table>
<thead>
<tr>
<th>proprietary</th>
<th>NVidia® CUDA</th>
<th>NVidia OpenACC</th>
<th>Intel LEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>data parallel</td>
<td>offload</td>
<td>offload</td>
</tr>
<tr>
<td>target (perf.)</td>
<td>NVidia GPUs</td>
<td>NVidia GPUs</td>
<td>portable</td>
</tr>
<tr>
<td>alternative</td>
<td>OpenCL®</td>
<td>OpenMP 4.0</td>
<td>OpenMP 4.0</td>
</tr>
</tbody>
</table>

© 2014, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>Non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP®</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>Implemented</td>
<td>Vendor libraries</td>
<td>Many</td>
<td>In compiler</td>
<td>Portable</td>
<td>In compiler</td>
</tr>
<tr>
<td>Supported by</td>
<td>Most vendors</td>
<td>Open src &amp; vendors</td>
<td>Most compilers</td>
<td>Ported most everywhere</td>
<td>Gcc and Intel (llvm future)</td>
</tr>
</tbody>
</table>

Compare...

<table>
<thead>
<tr>
<th>Proprietary</th>
<th>NVidia CUDA</th>
<th>NVidia OpenACC</th>
<th>Intel LEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Data parallel</td>
<td>Offload</td>
<td>Offload</td>
</tr>
<tr>
<td>Target (perf.)</td>
<td>NVidia GPUs</td>
<td>NVidia GPUs</td>
<td>Portable</td>
</tr>
<tr>
<td>Alternative</td>
<td>OpenCL</td>
<td>OpenMP 4.0</td>
<td>OpenMP 4.0</td>
</tr>
</tbody>
</table>
Choosing a non-proprietary parallel abstraction

<table>
<thead>
<tr>
<th>non-proprietary</th>
<th>BLAS, FFTW</th>
<th>MPI</th>
<th>OpenMP&lt;sup&gt;®&lt;/sup&gt;</th>
<th>TBB</th>
<th>Cilk™ Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>prog. lang.</td>
<td>Fortran, C, C++</td>
<td>Fortran, C, C++</td>
<td>Fortran or C</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>implemented</td>
<td>vendor libraries</td>
<td>many</td>
<td>in compiler</td>
<td>portable</td>
<td>in compiler</td>
</tr>
<tr>
<td>supported by</td>
<td>most vendors</td>
<td>open src &amp; vendors</td>
<td>most compilers</td>
<td>ported most everywhere</td>
<td>gcc and Intel (llvm future)</td>
</tr>
<tr>
<td>composable?</td>
<td>usually</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>memory</td>
<td>shared/distributed</td>
<td>distributed</td>
<td>shared (in implementations)</td>
<td>shared memory</td>
<td>shared memory</td>
</tr>
<tr>
<td>tasks</td>
<td>yes</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>limited keywords, TBB</td>
</tr>
<tr>
<td>explicit SIMD</td>
<td>Internal</td>
<td>n/a</td>
<td>YES (OpenMP 4.0: SIMD)</td>
<td>use compiler options, OpenMP directives, or Cilk Plus keywords</td>
<td>keywords</td>
</tr>
<tr>
<td>offload</td>
<td>some</td>
<td>n/a</td>
<td>YES (OpenMP 4.0: SIMD)</td>
<td>use Cilk Plus or OpenMP</td>
<td>keywords</td>
</tr>
</tbody>
</table>
It’s your Forest

Increase exposing parallelism.
Increase locality of reference.

YOUR MISSION
Questions?

james.r.reinders@intel.com