High Performance Computing, @ Intel: Do we have universal solution?

Paweł Gepner

HPC Platform Architecture Specialist
Intel Corporation

What HPC Users Want

- **High Performance on <u>my</u> code**
  - High Floating Point performance
  - High Integer Performance
  - Enough memory bandwidth
  - Enough network bandwidth and latency
  - Enough I/O bandwidth and latency
  - Enough Memory
- **Enough Reliability**
  - Days
  - Minutes is not enough
  - Months is too expensive
  - Mitigated by check-pointing

Total Application Performance

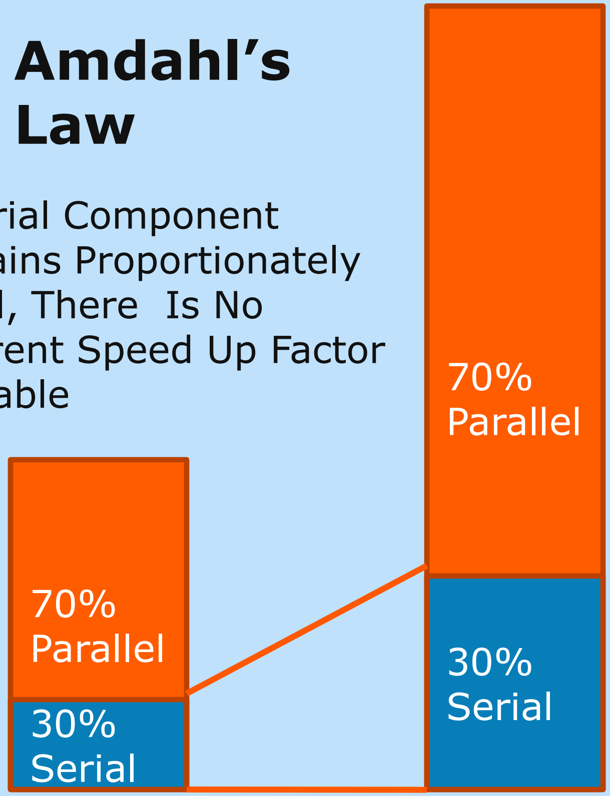
$$T_{\text{solution}} = T_{\text{compute}} + T_{\text{MEM B/W}} + T_{\text{I/O}} + T_{\text{node comms}}$$

Kernel benchmarking has its place, but benchmarking total application performance is critical for accurately evaluating HPC systems.

The Truth Of Law's & Observations

Amdahl's Law

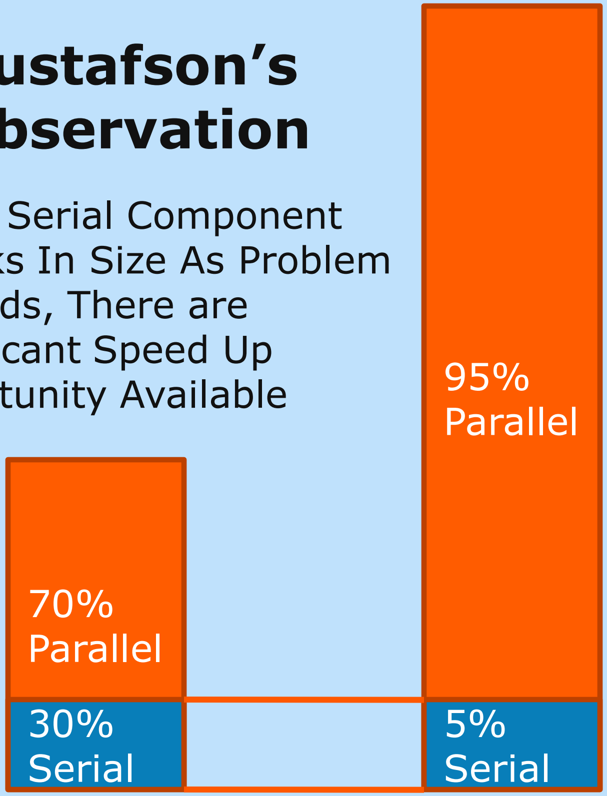
If Serial Component Remains Proportionately Equal, There Is No Inherent Speed Up Factor Available



If parallel component is 200x faster the max speed up is 3.25X

Gustafson's Observation

If The Serial Component Shrinks In Size As Problem Expands, There are Significant Speed Up Opportunity Available



If parallel component is 200x faster the max speed up is 18.26X

Growing the problem size may mitigate the impact of Amdahl's Law.

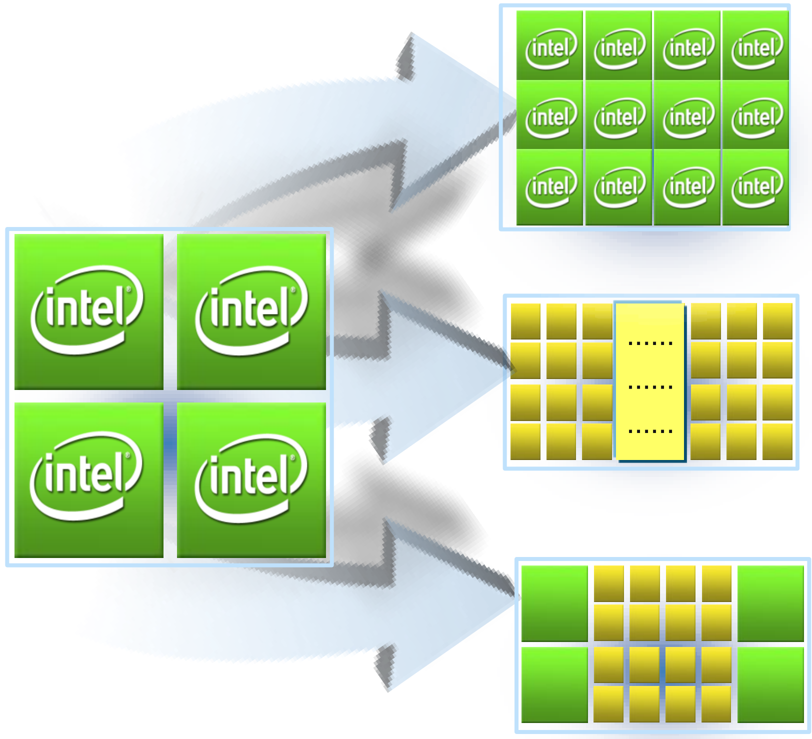
ONLY **if the serial fraction doesn't grow in proportion to the problem size**

Significant Shift In Architectures

Multiple Paths

To Increase

Performance

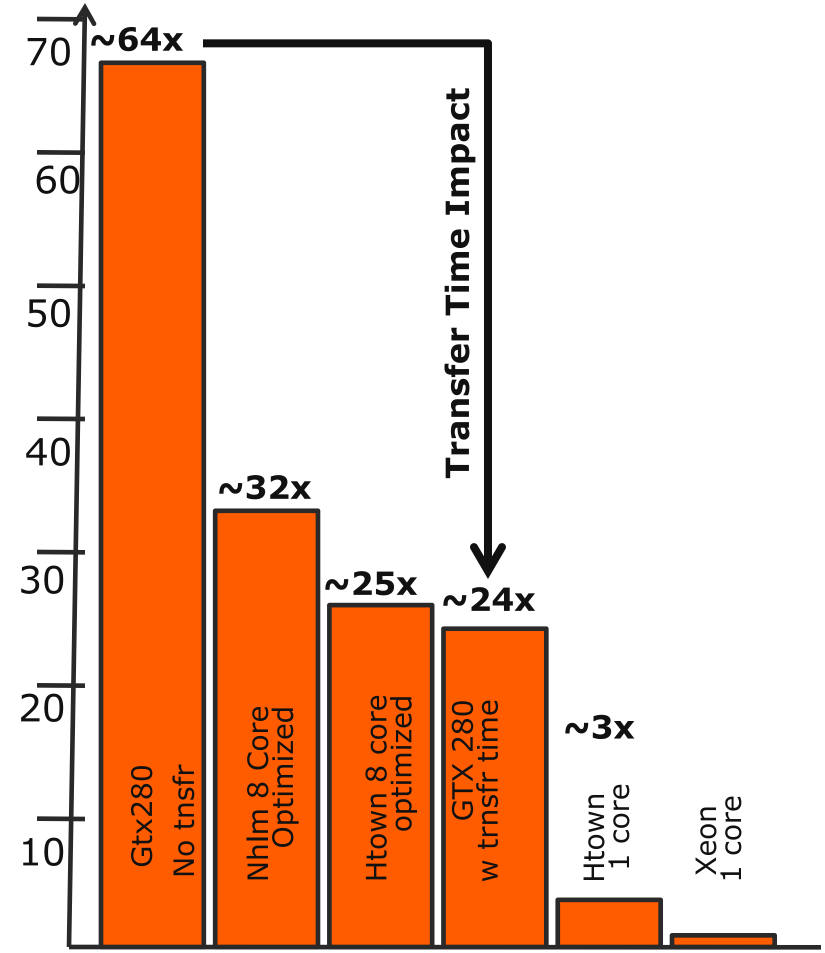


Why Accelerators

- Accelerators are not a new phenomenon:
  - in the 1980's
- HPC users never tend to be content, they:
  - Always want more performance than the they have systems at their disposal
  - Are continuously looking for ways to speed up their calculations or parts of them.
- Accelerators
  - May speed up some specialized computations, but may not be able to perform others.

Weather Modeling

- **Background**
  - WRF is a highly respected/referenced NCAR weather simulation modeling tool (WSM5) workload
  - Initial test were done against single core on an older 3.0GHz Xeon
  - Two form of optimization have occurred
- **Key Learning**
  - Compare new processors NOT old processor's
  - Insure the software is optimized
  - Focus on TOTAL time NOT kernel time
- **Results**
  - Intel current and next gen silicon delivers scalable performance



Optimizing Software On Traditional Intel Platforms Yields Startling Results

Paths To Accelerated Performance

Intel Approach:	Point Accelerators (GPU, Cell)
Support Data, Thread, Process, & Machine Level parallelism	Optimize primarily data parallel
Evolutionary – Scales Up, then out over time	Point design: optimized to solve a specific problem
Scale Forward existing software	Custom write and optimize application to the machine generation

Both Solutions Work

Accelerators-What do they look like?

- The scene is roughly divided in three unequal parts:

  1. Graphical cards or Graphical Processing Units
  (GPUs as opposed to the general CPUs).
  - ATI Firestream 9170
  - Nvidia C1060

for good performance one needs knowledge of the memory structure on the card to exploit it accordingly

  2. General computational floating-point accelerators.
  - Clearspeed

ClearSpeed processors were made to operate on 64-bit floating-point data from the start and possess that capability to process full error correction is present in the ClearSpeed processors

  - IBM Cell

  Cell Broadband Engine (Cell BE), was designed at least partly with the gaming industry in mind. The PowerXCell 8i won its share of fame for it's use in the Roadrunner system at Los Alamos National Laboratory

  3. Field Programmable Gate Arrays
  - Alltera, DRC, Naltech, PICO, SRC…

  An FPGA (Field Programmable Gate Array) is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks.

Not all applications can benefit from accelerators.
Of those which can, not all may be accelerated to the same degree.

FPGA Accelerators – Pros & Cons

PROS

- Architecture Flexibility
- Very good solution for the same statics application
- Power – perf/watt

CONS

- Complex to develop
- FPGA clock is 10-20x slover then modern CPUs
- New skill set
- Tough to deploy in an IT environment
- Cost – Expensive HW & SW manpower

Cell – Pros & Cons

PROS

- High Volume Part – Decreased Cost
- Works if the code can fit in the memory

CONS

- Complex to develop
- Optimized code is not human readable.
- Work has to be split into 3 parts (SPE, PPC, X86) and has to map nicely to SPE structure and memory to work - complex

NVIDIA GP-GPU Pros & Cons

PROS

- Easy to learn, code still looks like C
- Just an add-on to a regular system
- CUDA evolution

CONS

- Difficult to achieve performance
- Only for data parallel applications
- Limited amount of memory/size of job
- Difficult to debug
- Harder to profile

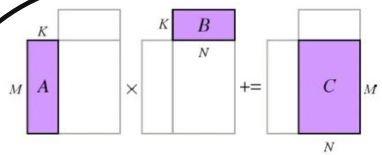
What's Intel's position on accelerators?

- Accelerators are an important component of any high performance computing strategy.
- Accelerators must support a standard programming model like OpenMP, unlike the specialized GPUs available today.
- The current GPGPU model today is limited to data parallel algorithms that require huge amounts of concurrency (e.g.32*240=7,680 threads).
- We believe an x86 based accelerator with its more robust instruction set can be an excellent accelerator platform with broader utility. This is in contrast to some accelerators which appear to be no more of an array of ALU's and hence very good at graphics.
- If a customer's applications fits the current GPGPU model description and they can fit in the GPU cards memory, that's great; they should use a GPU.
  - However, we believe the number of people outside of graphics that fall into that category is small.
- Users must consider the whole application, not just a kernel.
  - Many GPGPU papers talk about DGEMM or an FFT. *These only represent a fraction of the full application.*
  - How many full applications are there "out there' that can keep thousands of threads busy? Not very many.

Intel’s position on accelerators (contd)

- Amdahl’s law is real.
  - The serial fraction of an application will limit the available speedup (e.g. if 70% of an application in considerably fast (0 seconds), then largest possible speedup factor is 3.25X).
- GPGPU programming can be very complicated.
  - You cannot move MPI and OpenMP programs onto a GPU without rewriting them in a GPGPU language.
  - The most commonly referenced GPU language, CUDA is significantly more difficult to use than OpenMP or MPI.
  - CUDA locks you into Nvidia. Intel believes in open standards based tools that create portability. We see CUDA as unacceptable and very proprietary.
  - We have worked with Apple, Nvidia, IBM, AMD, TI and others to define and support a portable GPGPU language called OpenCL.
  - The industry standard specification for OpenCL is scheduled for completion in December 2008.
- With a single source code base, you’ll be able to support GPUs and CPUs … and from multiple vendors. Much better than CUDA.

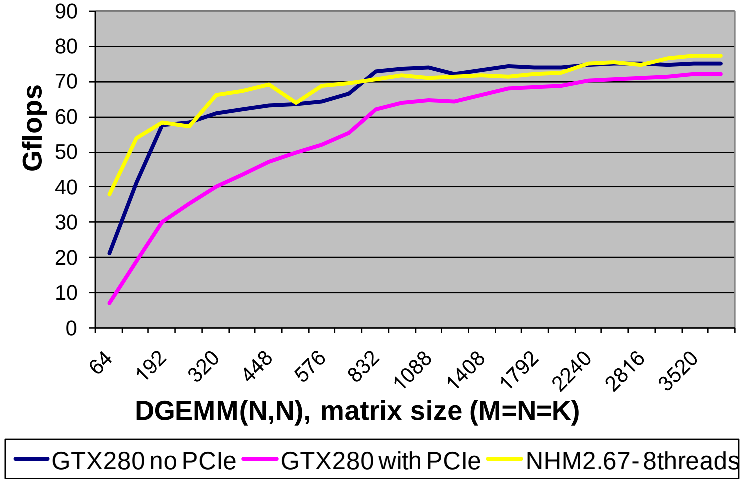
Scale it Forward – DGEMM Kernel

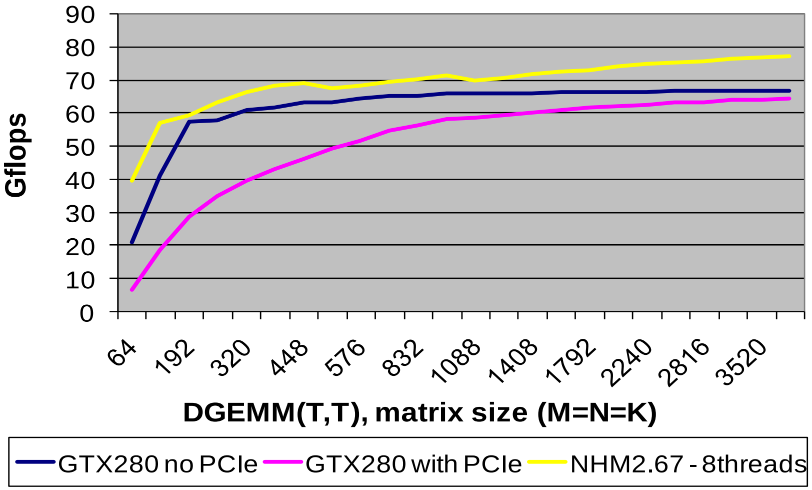


DGEMM (MKL)

Matrix-Matrix Multiply

- Used in dense and sparse linear equation solvers, MCAE, Computational chemistry
- 90% of computation time of LINPACK* (TOP500* list) spent in this kernel





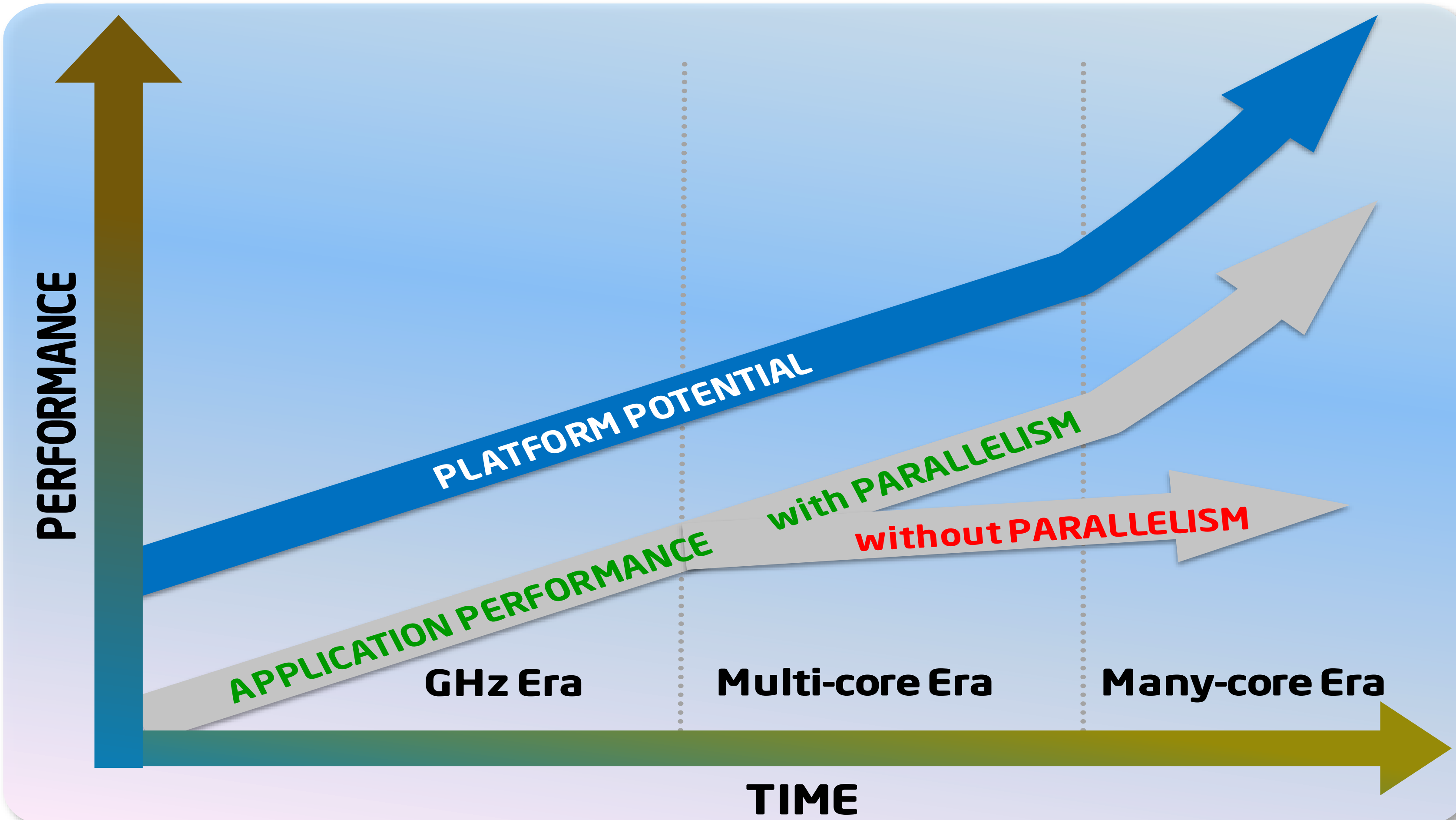
Next generation Intel® Xeon 5500 competitive DGEMM Perf

w/PCIe includes Total time: Data in, Compute, Data Out
No PCIe: Kernel time only, Compute

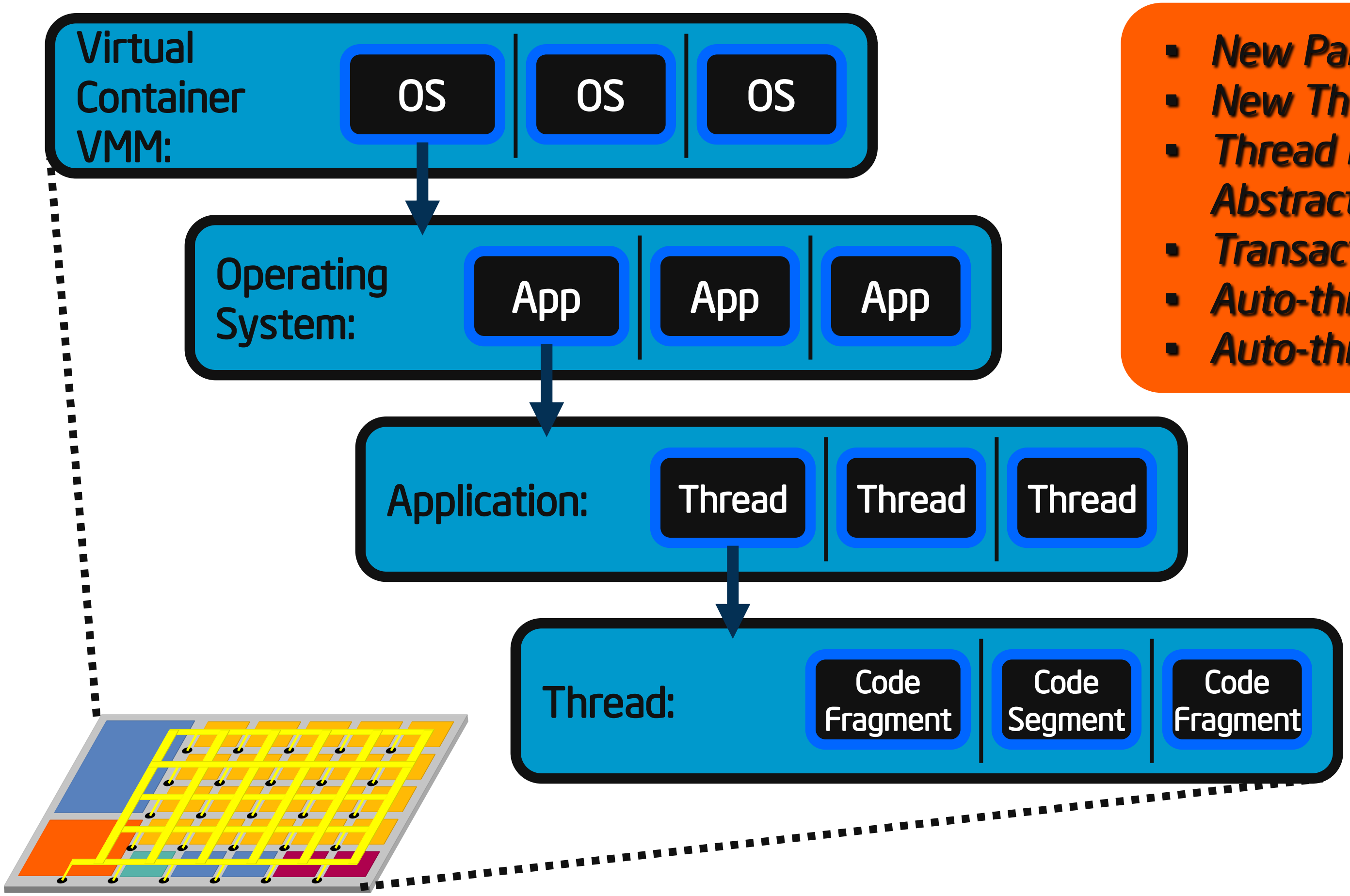
[1] Next Generation Intel® microarchitecture (Nehalem) over 1C Intel® Core™ 2 Quad Proc

See backup to configuration details

Multi-/Many-Core Era Imposed a Strategic Inflection Point For Parallelism

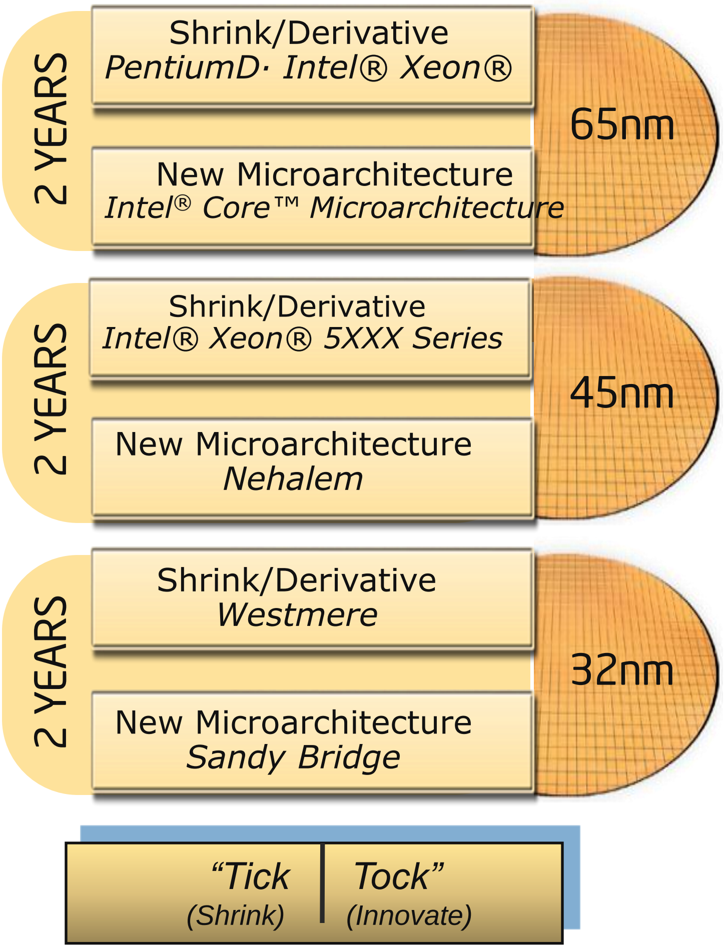


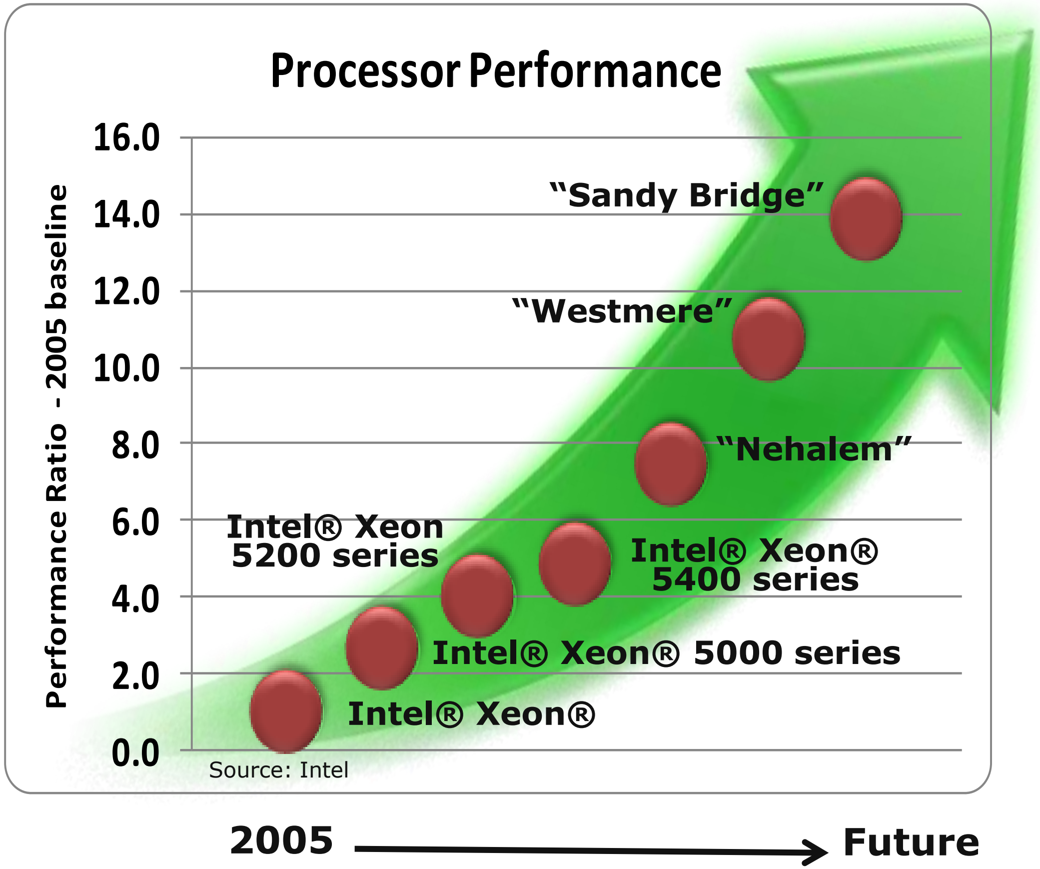
Parallelism at many levels



- *New Parallel Languages*
- *New Threading tools*
- *Thread Management & Abstraction layers*
- *Transactional memory*
- *Auto-threading compilers*
- *Auto-threading hardware*

Delivering Leadership Multi-Core Performance





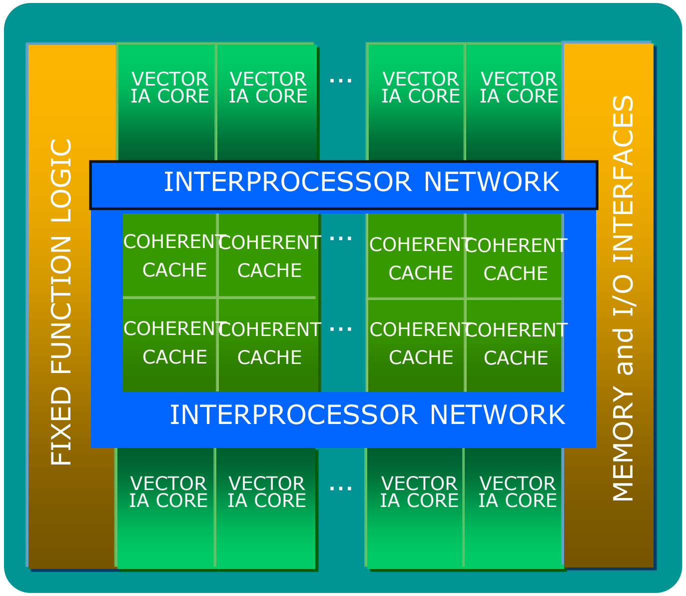
Silicon and Software Tools Unleash Performance

Instruction Extensions : Intel® AVX

A 256-bit vector extension to SSE that benefits floating point intensive applications

KEY FEATURES	BENEFITS
Wider Vectors Increased from 128 bit to 256 bit	Up to 2x peak FLOP output (floating point operations per second)
Enhanced Data Rearrangement Use the new 256 bit primitives to broadcast, mask loads and do data permutes	Organize, access and pull only necessary data more quickly and efficiently
Three Operand, Non Destructive Syntax Designed for efficiency and future extensibility	Fewer register copies, better register use, more opportunities for parallel loads and compute operations, smaller code size

Throughput Computing:
Accelerating Your Discovery



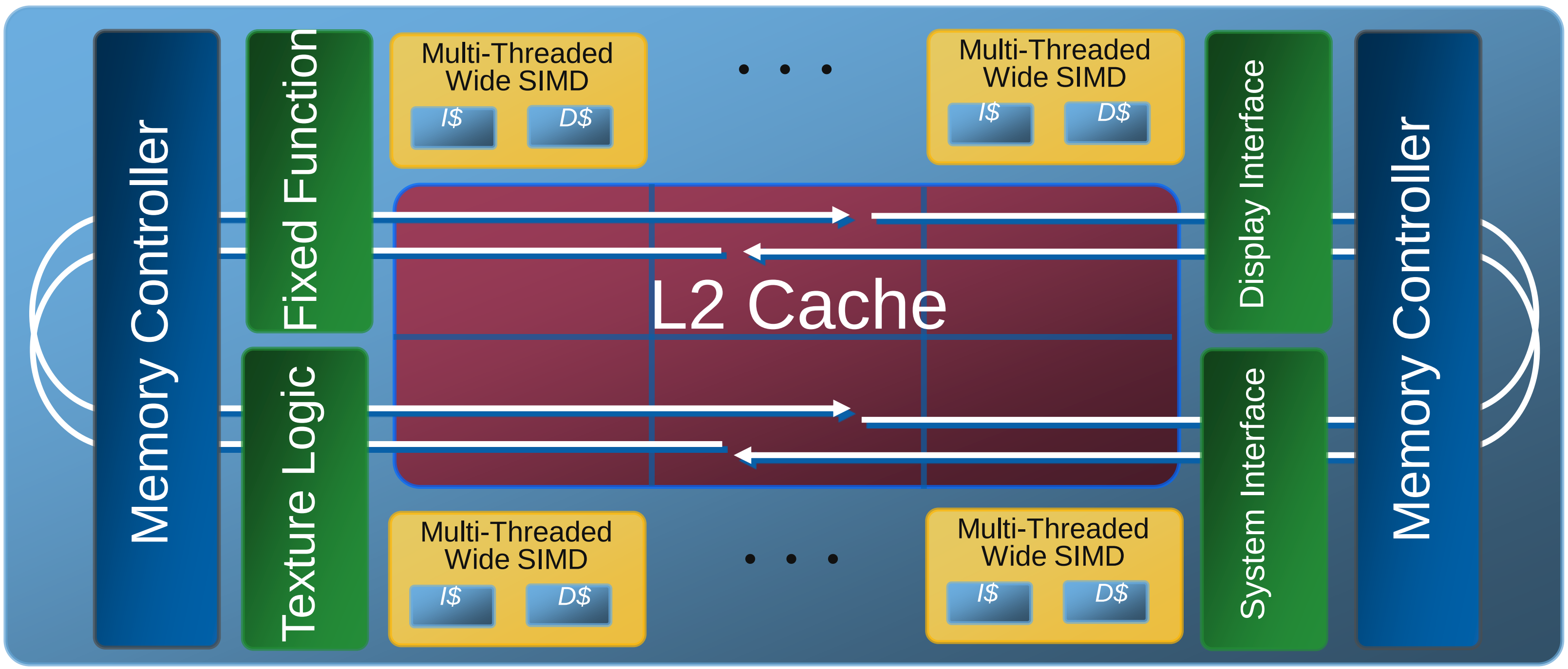
Intel Many Core Breakthroughs

- **Array Of Fully Programmable IA Cores**
- **Innovative Hardware Caching Architecture**
- **Scales To Tera FLOPS**
- **Single Development Environment**
- **Single Software Executable**

Best Architecture for the Best Algorithms

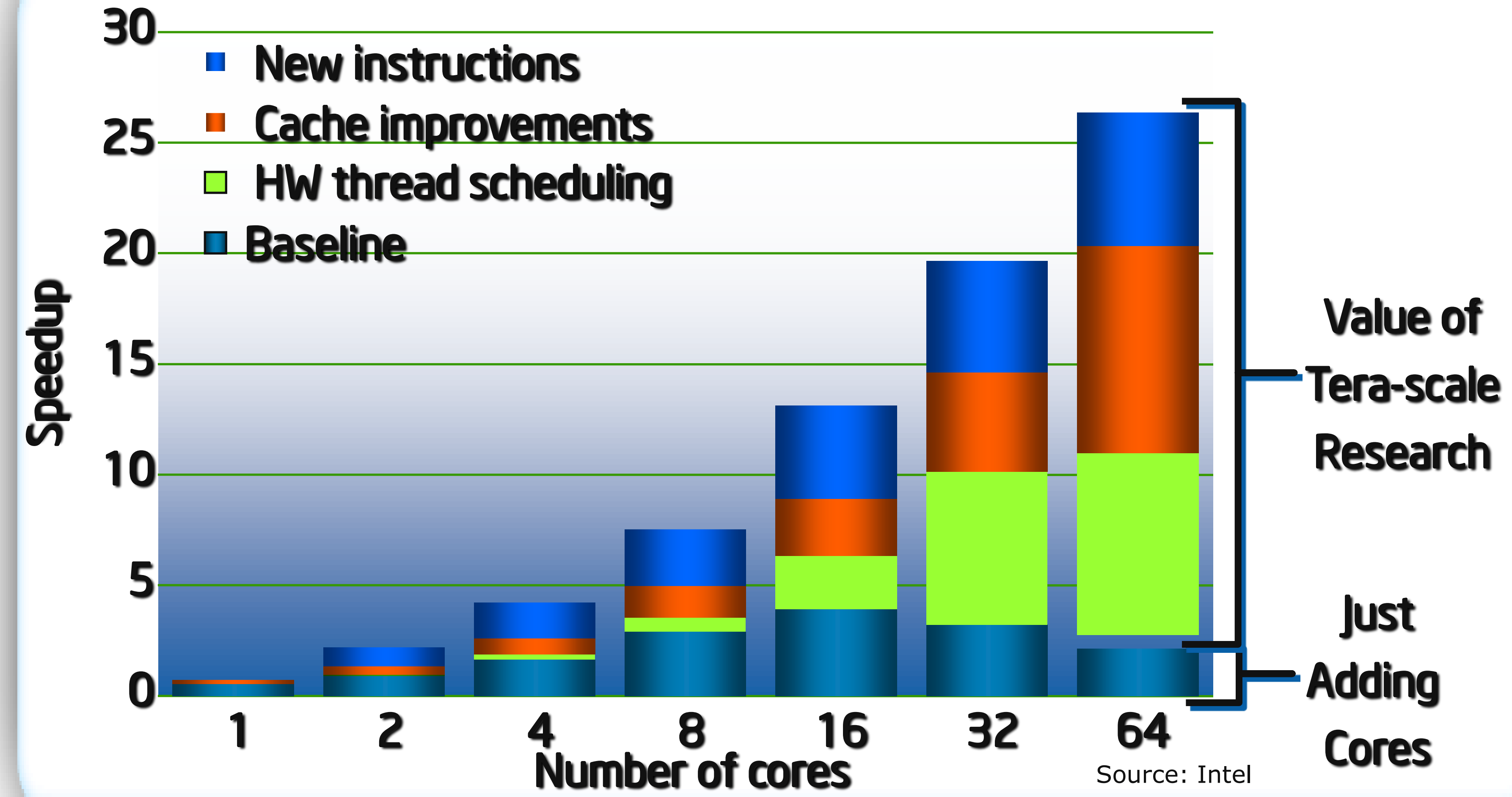
In The Same Programming Environment

Larrabee Block Diagram



- Cores communicate on a wide ring bus
  - Fast access to memory and fixed function blocks
  - Fast access for cache coherency
- L2 cache is partitioned among the cores
  - Provides high aggregate bandwidth
  - Allows data replication & sharing

PUTTING IT ALL TOGETHER



The Challenge Parallel Programming

Irregular Patterns, Data Structures and Serial Algorithms

Scale to Multi-Core Today → Hard

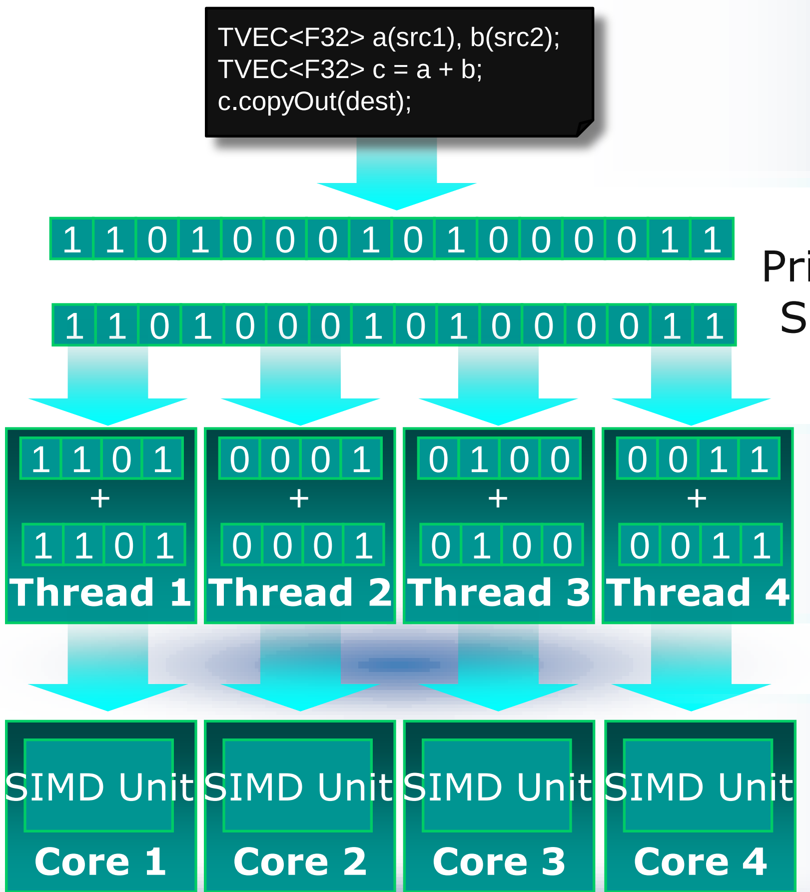
Scale to Many-Core Tomorrow → Harder

Increasing Cores (2→64+ Cores)

Vector Instructions (4→8+ Wide)

Cache and Interconnect Latency

Ct: A Throughput Programming Language



User Writes Serial-Like
Core Independent C++ Code

Primary Data Abstraction is the Nested Vector
Supports Dense, Sparse, and Irregular Data

Ct Parallel Runtime:
Auto-Scale to Increasing Cores

Ct JIT Compiler:
Auto-vectorization,
SSE, AVX, Larrabee

Programmer Thinks Serially; Ct Exploits Parallelism

Summary

- **Achieve Breakthrough Performance**
  - Optimize SW once for multi/many core solutions that scales forward
  - Employ tools that scale forward to multi and many core architectures
  - Special case need special Hardware
- **Maximize TCO**
  - Employ a single software development environment
  - Leverage existing software investments.