

Programming Larrabee: Beyond Data Parallelism

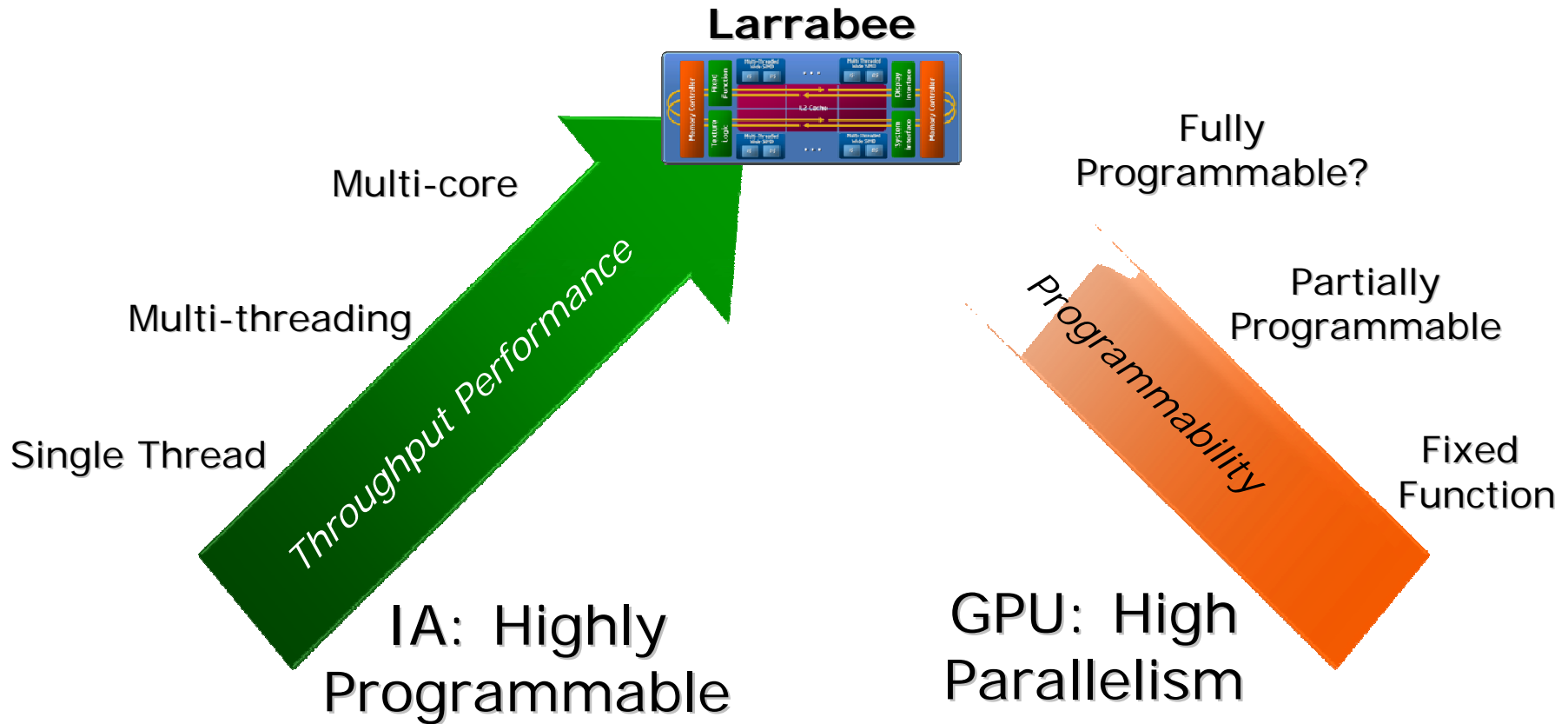


Dr. Larry Seiler
Intel Corporation

Agenda:

- What do parallel applications need?
- How does Larrabee enable parallelism?
- How does Larrabee support data parallelism?
- How does Larrabee support task parallelism?
- How can applications mix parallel methods?

Architecture Convergence



CPU programmability & GPU parallelism

Programming Larrabee

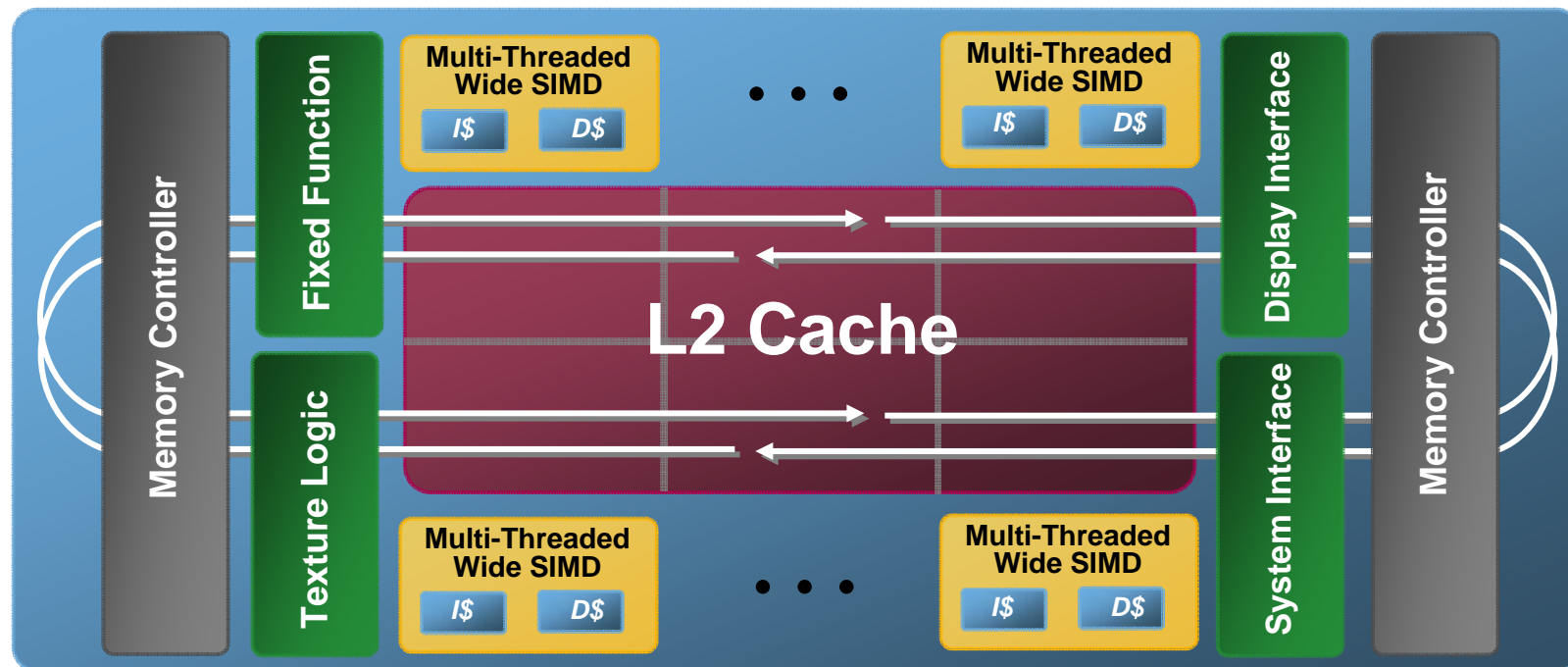
- Larrabee supports “braided parallelism”:
 - GPU-style data-parallelism, mixed with
 - Multi-core task parallelism , mixed with
 - Streaming pipe parallelism , mixed with
 - Sequential code (still often necessary)
- Larrabee can submit work to itself
- Larrabee supports the IA architecture

*Larrabee lets applications mix
the kinds of parallelism they need*

Agenda

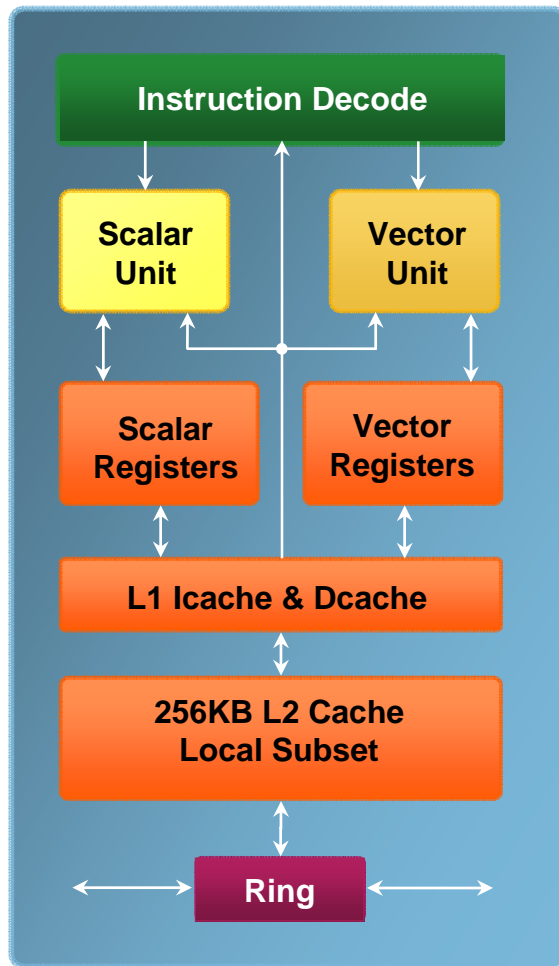
- What do parallel applications need?
- How does Larrabee enable parallelism?
- How does Larrabee support data parallelism?
- How does Larrabee support task parallelism?
- How can applications mix parallel methods?

Larrabee Block Diagram



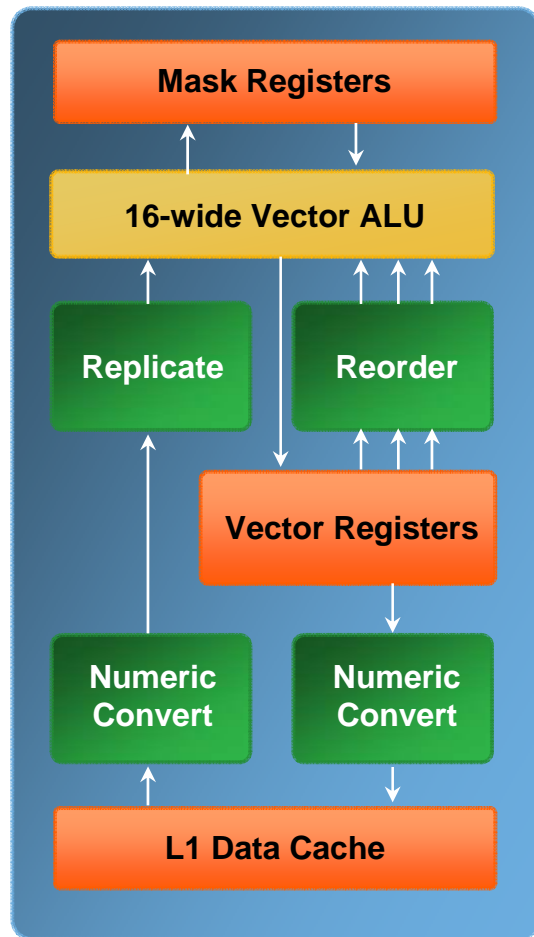
- Cores communicate on a wide ring bus
 - Fast access to memory and fixed function blocks
 - Fast access for cache coherency
- L2 cache is partitioned among the cores
 - Provides high aggregate bandwidth
 - Allows data replication & sharing

Processor Core Block Diagram



- Separate scalar and vector units with separate registers
- Vector unit: 16 32-bit ops/clock
- In-order instruction execution
- Short execution pipelines
- Fast access from L1 cache
- Direct connection to each core's subset of the L2 cache
- Prefetch instructions load L1 and L2 caches

Vector Unit Block Diagram



- Vector complete instruction set
 - Scatter/gather for vector load/store
 - Mask registers select lanes to write, which allows data-parallel flow control
 - Masks also support data compaction
- Vector instructions support
 - Memory operand for most instructions: full speed when data in L1 cache
 - Free numeric type conversion and data replication while reading from memory
 - Rearrange the lanes on register read
 - Fused multiply add (three arguments)
 - Int32, Float32 and Float64 data

Key Differences from Typical GPUs

- Each Larrabee core is a complete Intel processor
 - Context switching & pre-emptive multi-tasking
 - Virtual memory and page swapping, even in texture logic
 - Fully coherent caches at all levels of the hierarchy
- Efficient inter-block communication
 - Ring bus for full inter-processor communication
 - Low latency high bandwidth L1 and L2 caches
 - Fast synchronization between cores and caches

*Larrabee: the programmability of IA
with the parallelism of graphics processors*

Agenda

- What do parallel applications need?
- How does Larrabee enable parallelism?
- How does Larrabee support data parallelism?
- How does Larrabee support task parallelism?
- How can applications mix parallel methods?

Data Parallelism Overview

- Key ideas of data parallel programming
 - Define an array (grid) of data elements
 - Run same program (kernel) over all the elements
 - Share memory between groups within the grid
 - Basically just nested parallel “for” loops
- Strengths
 - Good utilization of SIMD processing elements (VPU)
 - Hides long latencies (memory or texture accesses)
- Limitations
 - Works best for large, homogenous problems
 - Work efficiency drops with irregular conditional execution
 - Work efficiency drops with irregular data structures

Larrabee Data Parallelism: Strands

Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand

16-wide vector unit

...

Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand

16-wide vector unit

Each strand is the kernel running on one data element

Each strand maps onto one (or more) VPU lanes

Strands must support different conditional execution

Larrabee Data Parallelism: Fibers

Fiber: SW-managed context (hides long predictable latencies)

Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand

16-wide vector unit

...

Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand
Strand

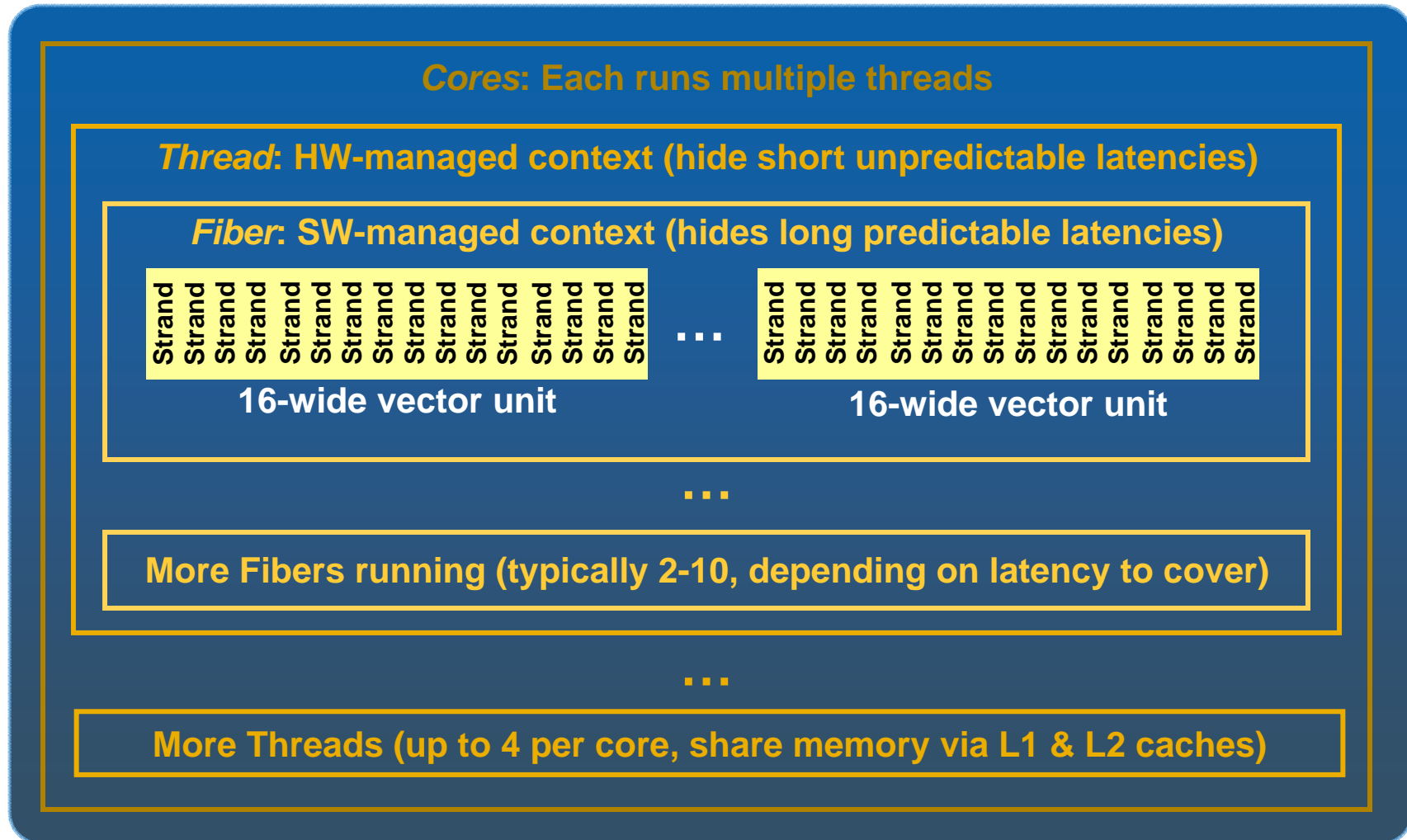
16-wide vector unit

...

More Fibers running (typically 2-10, depending on latency to cover)

SW cycles among fibers groups of strands) to cover latency

Larrabee Data Parallelism: Threads



Data Parallelism Summary

- Implementation hierarchy (your names may vary)
 - Strand: runs a data kernel in one (or more) VPU lane(s)
 - Fiber: SW-managed group of strands, like co-routines
 - Thread: HW-managed, swaps fibers to cover long latencies
 - Core: runs multiple threads to cover short latencies
 - Note: we use “thread” and “core” in micro-processor sense
- Comparison to GPU data parallelism
 - Same mechanisms as used in GPUs, except...
 - Larrabee allows SW scheduling (except for HW threads)

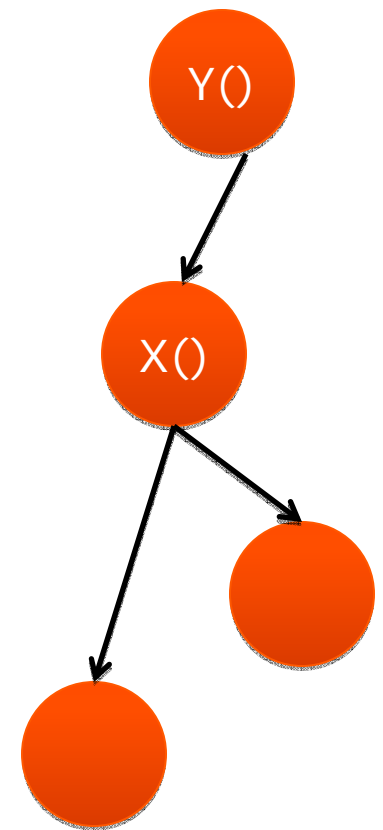
***Larrabee: uses many-core IA architecture
to execute data parallel workloads***

Agenda

- What do parallel applications need?
- How does Larrabee enable parallelism?
- How does Larrabee support data parallelism?
- How does Larrabee support task parallelism?
- How can applications mix parallel methods?

Task Parallelism Overview

- Think of a task as an asynchronous function call
 - “Do X at some point in the future”
 - Optionally “... after Y is done”
 - Light weight, often in user space
- Strengths
 - Copes well with heterogeneous workloads
 - Doesn't require 1000's of strands
 - Scales well with core count
- Limitations
 - No automatic support for latency hiding
 - Must explicitly write SIMD code



Larrabee Task Parallelism: Tasks

Task: SW-managed context

Scalar code

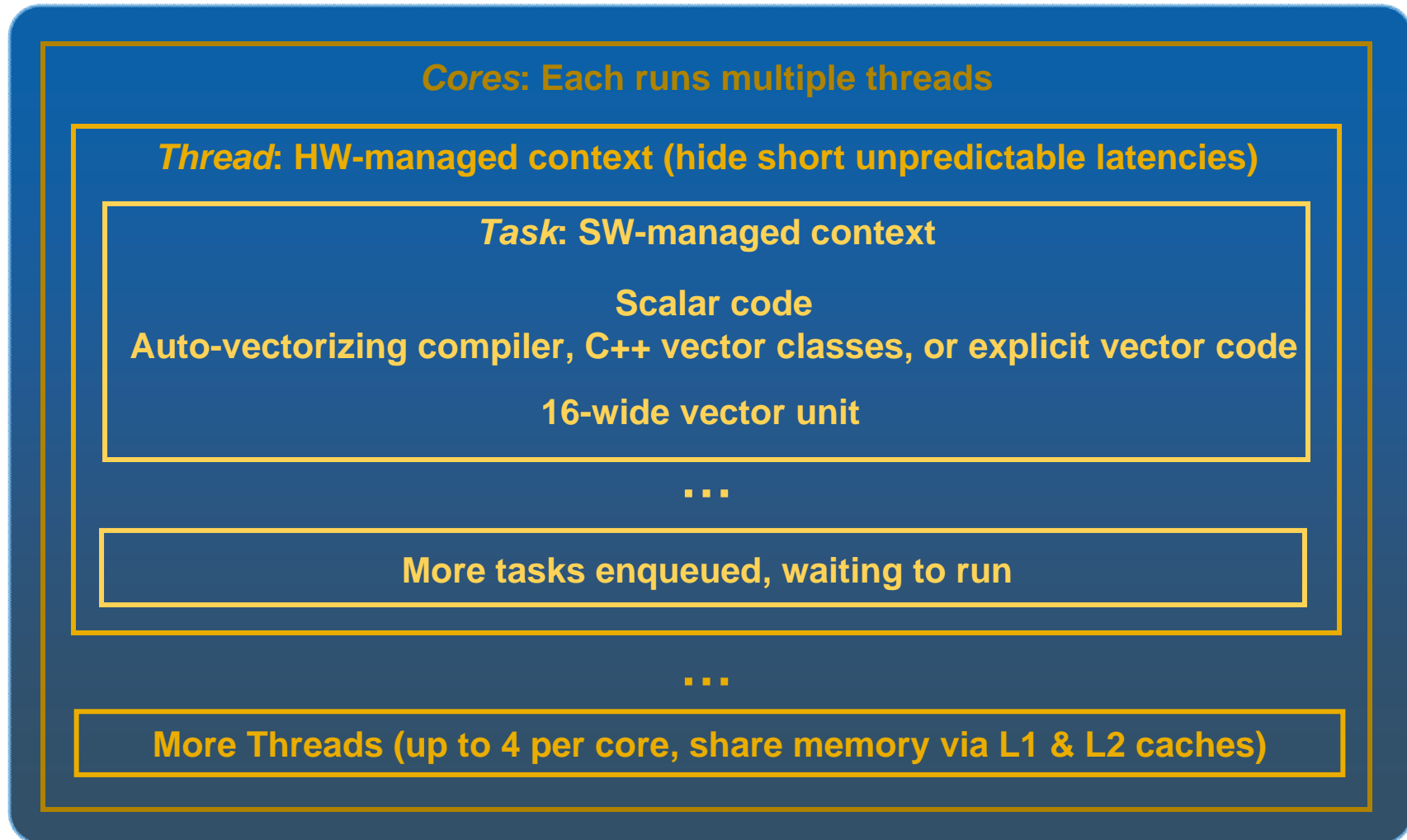
Auto-vectorizing compiler, C++ vector classes, or vector intrinsics

16-wide vector unit

...

More tasks enqueued, waiting to run

Larrabee Task Parallelism: Threads



Examples of Using Tasks

- Applications
 - Scene traversal and culling
 - Procedural geometry synthesis
 - Physics contact group solve
 - Data parallel strand groups
 - Distribute across threads/cores using task system
 - Exploit core resources with fibering/SIMD
- Larrabee can submit work to itself!
 - Tasks can spawn other tasks
 - Exposed in Larrabee Native programming interface

Larrabee Native Programming

- Internal projects include tools and compilers
- C and C++:
 - std C libs: printf(), malloc(), etc.
 - Data pointers, function pointers, recursion, etc.
- Threading: spawn P-Threads or use tasks
- Full software control when needed
 - Thread affinity, assembly code, etc.
 - Access to fixed function units

*Larrabee: supports fully
general purpose task parallel programming*

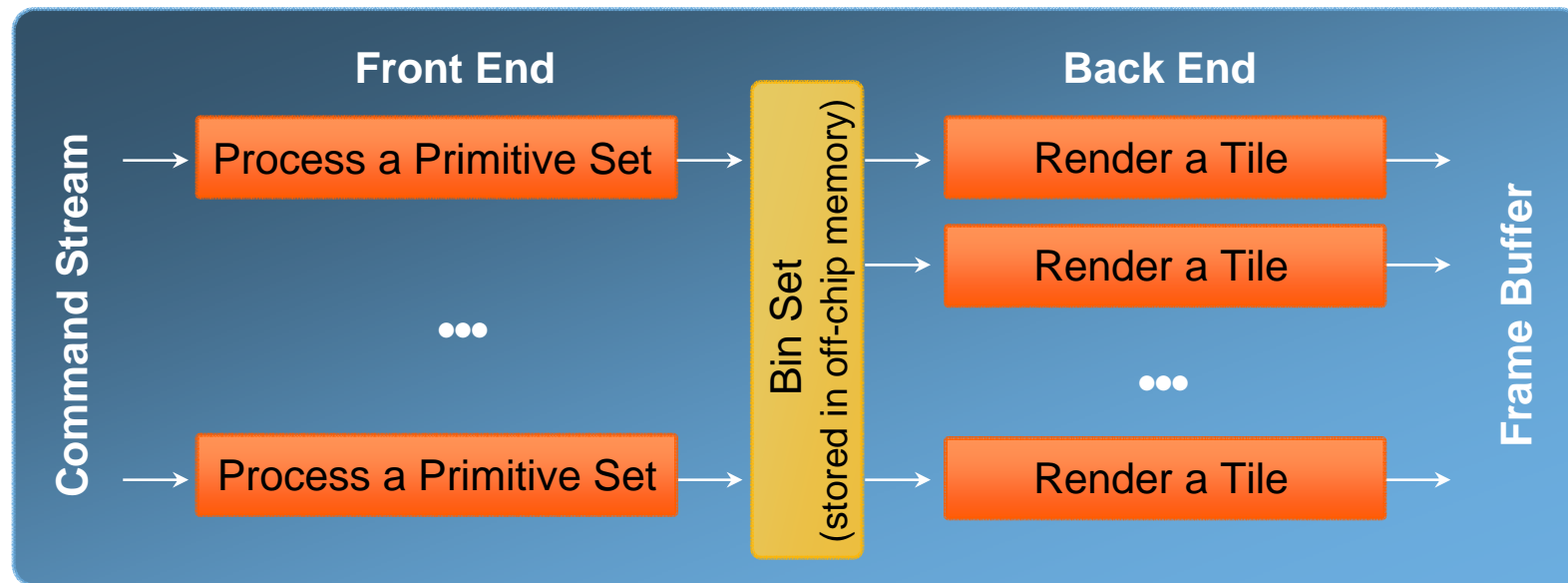
Agenda

- What do parallel applications need?
- How does Larrabee enable parallelism?
- How does Larrabee support data parallelism?
- How does Larrabee support task parallelism?
- How can applications mix parallel methods?

Applications Can Be Complex

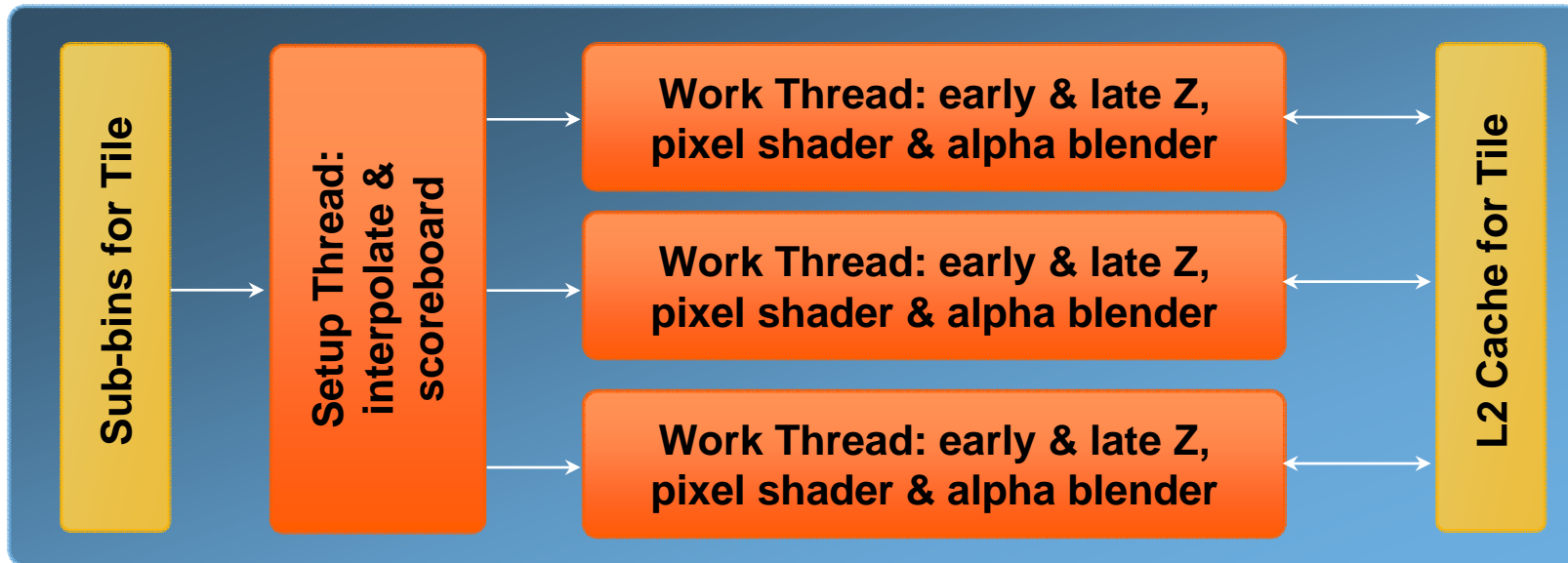
- May need many kinds of parallel code
 - GPU-style data-parallelism
 - Multi-core task parallelism
 - Streaming pipe parallelism
 - And even some sequential code
- May also need
 - Self-scheduling (data-dependent new work)
 - Irregular data structures
- Example? The rendering pipeline
 - All in software in Larrabee, except for texture filtering
 - Works due to support for irregular computation

Larrabee's Binning Renderer



- Binning pipeline
 - Reduces synchronization
 - Front end processes vertices
 - Back end processes pixels
 - Bin FIFO between them
- Multi-tasking by cores
 - Each orange box is a core
 - Cores run independently
 - Other cores can run other tasks, e.g. physics

Back-End Rendering a Tile

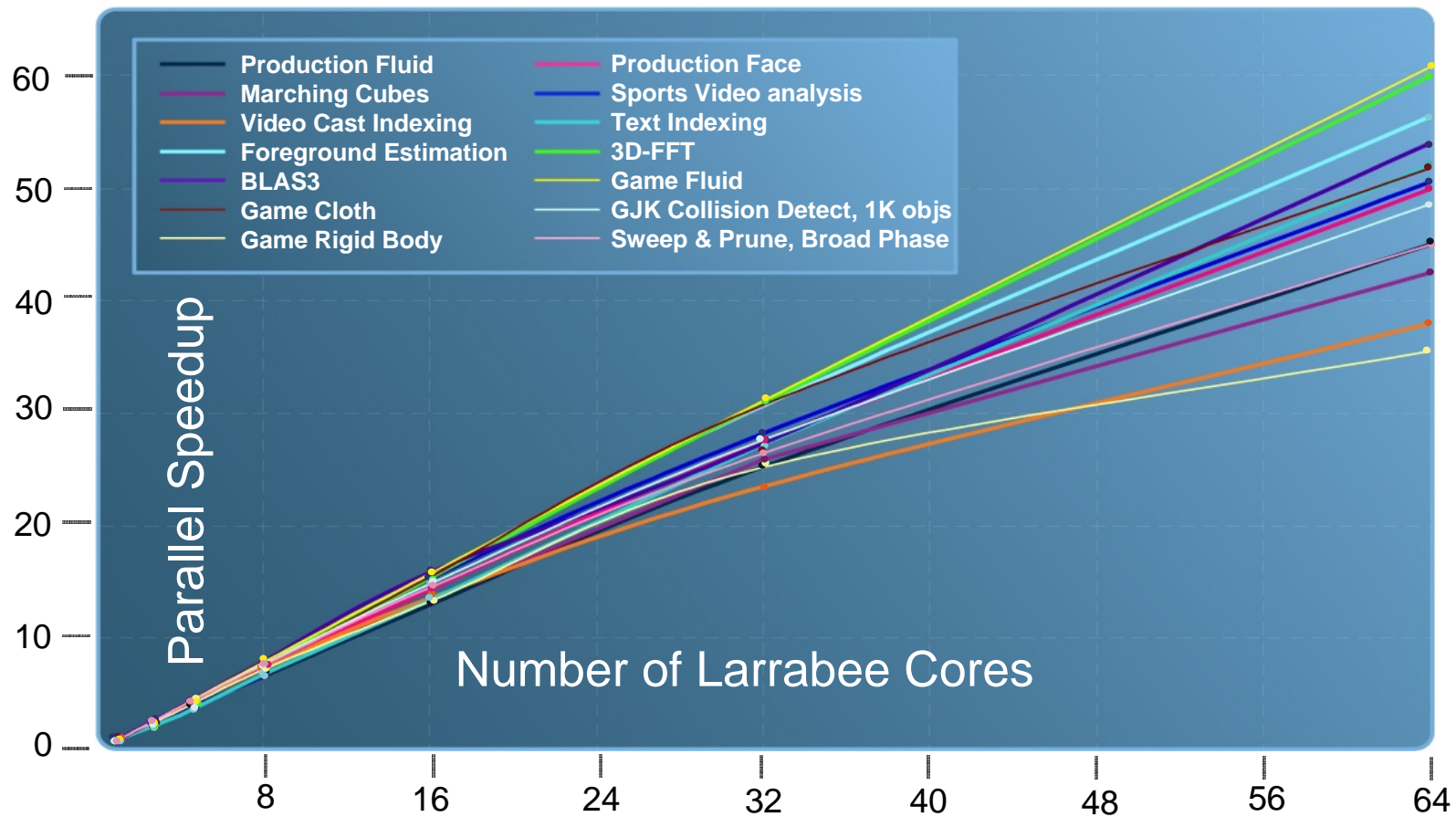


- Orange boxes represent work on separate HW threads
- Three work threads do Z, pixel shader, and blending
- Setup thread reads from bins and does pre-processing
- Combines task parallel, data parallel, and sequential

Pipeline Can Be Changed

- Parts can move between front end & back end
 - Vertex shading, tessellation, rasterization, etc.
 - Allows balancing computation vs. bandwidth
- New features can be added
 - Transparency, shadowing, ray tracing etc.
 - Each of these need irregular data structures
 - They benefit from mix of tasks and data-parallel
 - Also helps to be able to “repack” the data
- Graphics pipeline is just an example
 - These methods apply to many parallel applications

Application Scaling Studies



Data in graph from Seiler, L., Carmean, D., et al. 2008. *Larrabee: A many-core x86 architecture for visual computing*. SIGGRAPH '08: ACM SIGGRAPH 2008 Papers, ACM Press, New York, NY

“Choose the Right Tool”

- Use the 3D rendering pipeline
 - How you always have (or create your own)
- Use data parallelism
 - Parallel ‘for’ loop
 - Hide latency by switching to work from other elements
- Use task parallelism
 - Asynchronous function call
 - Hide latency with in-task reuse and asynch memory ops
- Use sequential code
 - Tie it all together
 - Communicate with host at higher level of abstraction

Flexible & programmable for many applications

Summary

- Larrabee lets applications mix the kinds of parallelism they need
- Larrabee provides the programmability of IA with the parallelism of GPUs
- Larrabee uses the many-core IA architecture to execute data parallel workloads
- Larrabee supports fully general purpose task parallel programming
- Larrabee is flexible & programmable for many applications

Larrabee: Beyond Data Parallelism

- Questions?

- See also

- s08.idav.ucdavis.edu for slides from a Siggraph2008 course titled "Beyond Programmable Shading"
- www.intel.com/idf -- click "content catalog" & search for "Larrabee:" to find a Larrabee presentation
- Seiler, L., Carmean, D., et al. 2008. *Larrabee: A many-core x86 architecture for visual computing*. SIGGRAPH '08: ACM SIGGRAPH 2008 Papers, ACM Press, New York, NY
- Fatahalian, K., Houston, M., "GPUs: a closer look", Communications of the ACM October 2008, vol 51 #10. graphics.stanford.edu/~kayvonf/papers/fatahalianCACM.pdf

Data in graph from Seiler, L., Carmean, D., et al. 2008. *Larrabee: A many-core x86 architecture for visual computing*. SIGGRAPH '08: ACM SIGGRAPH 2008 Papers, ACM Press, New York, NY

Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Larrabee and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel Inside and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright © 2009 Intel Corporation.

Risk Factors

This presentation contains forward-looking statements that involve a number of risks and uncertainties. These statements do not reflect the potential impact of any mergers, acquisitions, divestitures, investments or other similar transactions that may be completed in the future. The information presented is accurate only as of today's date and will not be updated. In addition to any factors discussed in the presentation, the important factors that could cause actual results to differ materially include the following: Demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including conditions in the credit market that could affect consumer confidence; customer acceptance of Intel's and competitors' products; changes in customer order patterns, including order cancellations; and changes in the level of inventory at customers. Intel's results could be affected by the timing of closing of acquisitions and divestitures. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of new Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel's response to such actions; Intel's ability to respond quickly to technological developments and to incorporate new features into its products; and the availability of sufficient supply of components from suppliers to meet demand. The gross margin percentage could vary significantly from expectations based on changes in revenue levels; product mix and pricing; capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; excess or obsolete inventory; manufacturing yields; changes in unit costs; impairments of long-lived assets, including manufacturing, assembly/test and intangible assets; and the timing and execution of the manufacturing ramp and associated costs, including start-up costs. Expenses, particularly certain marketing and compensation expenses, vary depending on the level of demand for Intel's products, the level of revenue and profits, and impairments of long-lived assets. Intel is in the midst of a structure and efficiency program that is resulting in several actions that could have an impact on expected expense levels and gross margin. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in the countries in which Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the report on Form 10-Q for the quarter ended June 28, 2008.