

ABORATORY

# Performance Modeling and Analysis

## **Samuel Williams**

**Computational Research Division** Lawrence Berkeley National Lab







ENCE BERKELEY NATIONAL LABORATORY

# Acknowledgements

- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.
- This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- Tuomas Koskela for his creation of the Roofline Advisor demo slides







NATIONAL LABORATORY

# Introduction to Performance Modeling





# Why Use Performance Models or Tools?

- Identify performance bottlenecks
- Motivate software optimizations
- **Determine when we're done optimizing** 
  - Assess performance relative to machine capabilities ٠
  - Motivate need for algorithmic changes •
- Predict performance on future machines / architectures
  - Sets realistic expectations on performance for future procurements
  - Used for HW/SW Co-Design to ensure future architectures are well-suited for the ٠ computational needs of today's applications.





# **Computational Complexity**

- Assume run time is correlated with the number of operations (e.g. FP ops)
- Users define parameterize their algorithms, solvers, kernels
- Count the number of operations as a function of those parameters
- Demonstrate run time is correlated with those parameters



```
#pragma omp parallel for
for(i=0;i<N;i++){
for(j=0;j<N;j++){
double Cij=0;
r(k=0;k<N;k++){
ij += A[i][k] * B[k][j];
][j] = sum;
```

GEMM: O(N<sup>3</sup>) complexity where N is the number of rows (equation

## Why did we depart from ideal scaling?



# Data <u>Movement Complexity</u>

6

- Assume run time is correlated with the amount of data accessed (or moved)
- Easy to calculate amount of data accessed... count array accesses
- Data moved is more complex as it requires understanding cache behavior...
  - Compulsory<sup>1</sup> data movement (array • sizes) is a good initial guess...
  - ... but needs refinement for the effects of finite cache capacities

<sup>1</sup>Hill et al, "Evaluating Associativity in CPU Caches", IEEE Trans. Comput., 1989.





# expensive... Performing Flop's, or

Data

O(N)

 $O(N^2)$ 

 $O(N^2)$ 

O(N)

# **Machine Balance and Arithmetic Intensity**

- Data movement and computation can operate at different rates
- We define machine balance as the ratio of...

Peak DP Flop/s Peak Bandwidth **Balance =** 

...and arithmetic intensity as the ratio of...

AI = Flop's Performed Data Moved







# **Computational Depth**

- Sequential CPUs incur latencies and overheads on memory discontinuities and function calls
- Parallel machines incur similar overheads on synchronization (shared memory), point-to-point communication, reductions, and broadcasts.
- Thus, we can classify algorithms by depth (max depth of the algorithm's dependency chain)





# **Distributed Memory Performance Modeling**

- In distributed memory, one communicates by sending messages between processors.
- Messaging time can be constrained by several components...
  - Overhead (CPU time to send/receive a message)
  - Latency (time message is in the network; can be hidden)
  - Message throughput (rate at which one can send small messages... messages/second)
  - Bandwidth (rate one can send large messages... GBytes/s)
- Bandwidths and latencies are further constrained by the interplay of network architecture and contention
- Distributed memory versions of our algorithms can be differently stressed by these components depending on N and P (#processors)





- Many different components can contribute to kernel run time.
- Some are characteristics of the application, some are characteristics of the machine, and some are both (memory access pattern + caches).

**#FP operations** Flop/s Cache data movement Cache GB/s DRAM data movement DRAM GB/s PCIe data movement PCIe bandwidth Depth OMP Overhead MPI Message Size Network Bandwidth MPI Send: Wait ratio Network Gap #MPI Wait's Network Latency



Can't think about all these terms all the time for every application...





Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

> Roofline **#FP operations** Flop/s Model Cache data movement Cache GB/s DRAM data movement DRAM GB/s PCIe data movement PCIe bandwidth Depth OMP Overhead MPI Message Size Network Bandwidth MPI Send: Wait ratio Network Gap #MPI Wait's Network Latency

Williams et al, "Roofline: An Insightful Visual Performance Model For Multicore Architectures",



Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

> **#FP operations** Flop/s Cache data movement Cache GB/s DRAM data movement DRAM GB/s PCIe data movement PCIe bandwidth Depth OMP Overhead MPI Message Size Network Bandwidth MPI Send: Wait ratio Network Gap LogP #MPI Wait's Network Latency



Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

> **#FP operations** Flop/s Cache data movement Cache GB/s DRAM data movement DRAM GB/s PCIe data movement PCIe bandwidth Depth OMP Overhead MPI Message Size Network Bandwidth MPI Send: Wait ratio Network Gap #MPI Wait's Network Latency

Alexandrov, et al, "LogGP: incorporating long messages into the LogP model - one step closer

towards a realistic model for parallel computation", SPAA, 1995.





Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

**#FP operations** Flop/s Cache data movement Cache GB/s DRAM data movement DRAM GR/P Depth OMP C Think about wise Size Network R, model of the solution of the solu PCIe data movement PCIe band LogCA MPI Message Size Network B MPI Send:Wait ratio Network Gan #MPI Wait's Network Latericy

Bin Altaf et al, "LogCA: A High-Level Performance Model for Hardware Accelerators", ISCA,







NATIONAL LABORATORY

# Introduction to the Roofine Mode





# **Performance Models / Simulators**

- Historically, many performance models and simulators tracked latencies to predict performance (i.e. counting cycles)
- The last two decades saw a number of latency-hiding techniques...
  - Out-of-order execution (hardware discovers parallelism to hide latency)
  - HW stream prefetching (hardware speculatively loads data) ٠
  - Massive thread parallelism (independent threads satisfy the latency-bandwidth product)
- Effective latency hiding has resulted in a shift from a latency-limited computing regime to a **throughput-limited computing regime**





# **Roofline Model**

- **Roofline Model** is a throughput-oriented performance model...
  - Tracks rates not times
  - Augmented with Little's Law (concurrency = latency\*bandwidth)
  - Independent of ISA and architecture (applies ulletto CPUs, GPUs, Google TPUs<sup>1</sup>, etc...)

BERKELEY LAB	OMPUTATIONAL F	ESEARCH STAFF RESI Ch = Research = Roofine
Perfor	mance and	d Algorithr
PERFORMANCE AND ALGORITHMS RESEARCH Auto-tuning BeBOP EDGAR HpGISAXS HPGMG Roofine SciDAC TOP500 Previous Projects	Recofice P and the second seco	Performance mance model used to bound the perfor processor architectures. Rather than a mance by combining locality, bandwidt ne the resultant Roofline figure in order films model is Arithmetic Intensity. Arith 5-1 vector-vector increment ( x[1]=vg] etht of the vector size. Conversely, to allocate cache architecture, the y of 0.104 <sup>st</sup> logN and would grow slo arthaps 2 flops per byte. Finally, BLA cky.
Facebook	0.1-1.0 flops SpMV BLAS1.2 Stencils (PD Lett	Typically < 2 flo Typically < 2 flo Typically < 2 flo Typically < 2 flo Typically < 2 flo FFTs, Spectral Me Methods 1) O( log

https://crd.lbl.gov/departments/computer-science/PAR/research/roofline





- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

**#FP ops / Peak GFlop/s #Bytes / Peak GB/s** Time = max <





- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



## 1 / Peak GFlop/s #Bytes / #FP ops / Peak GB/s Time = max **#FP ops**



- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



# Peak GFlop/s (#FP ops / #Bytes) \* Peak GB/s #FP ops Time



- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

Note, Arithmetic Intensity (AI) = Flops / Bytes (as presented to DRAM)





- Plot Roofline bound using Arithmetic Intensity as the x-axis
- Log-log scale makes it easy to doodle, extrapolate performance along Moore's Law, etc...
- Kernels with AI less than machine balance are ultimately DRAM bound (we'll refine this later...)





# **Roofline Example #1**

- Typical machine balance is 5-10 flops per byte...
  - 40-80 flops per double to exploit compute capability ٠
  - Artifact of technology and money ٠
  - Unlikely to improve ٠
- Consider STREAM Triad...

#pragma omp parallel for for(i=0;i<N;i++){</pre> Z[i] = X[i] + alpha\*Y[i];

- 2 flops per iteration ٠
- Transfer 24 bytes per iteration (read X[i], Y[i], write Z[i]) ٠
- AI = 0.083 flops per byte == Memory bound ٠





# **Roofline Example #2**

## Conversely, 7-point constant coefficient stencil...

- 7 flops •
- 8 memory references (7 reads, 1 store) per point ٠
- Cache can filter all but 1 read and 1 write per point ٠
- AI = 0.44 flops per byte == memory bound, •

```
but 5x the flop rate
```

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){</pre>
for(j=1;j<dim+1;j++){</pre>
for(i=1;i<dim+1;i++){</pre>
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk
                 + old[ijk-1
                 + old[ijk+1
                 + old[ijk-jStride]
                 + old[ijk+jStride]
                 + old[ijk-kStride]
                 + old[ijk+kStride];
}}
```





- Real processors have multiple levels of memory
  - Registers
  - L1, L2, L3 cache
  - MCDRAM/HBM (KNL/GPU device memory)
  - DDR (main memory)
  - NVRAM (non-volatile memory)
- Applications can have locality in each level
  - Unique data movements imply unique Al's
  - Moreover, each level will have a unique bandwidth



- Construct superposition of Rooflines...
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple Al's and multiple bounds (flops, L1, L2, ... DRAM)...
  - ... performance is bound by the • minimum





- Construct superposition of Rooflines...
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple Al's and multiple bounds (flops, L1, L2, ... DRAM)...
  - ... performance is bound by the minimum





- Construct superposition of Rooflines...
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple Al's and multiple bounds (flops, L1, L2, ... DRAM)...
  - ... performance is bound by the minimum





- Construct superposition of Rooflines...
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple Al's and multiple bounds (flops, L1, L2, ... DRAM)...
  - ... performance is bound by the minimum





# **NUMA Effects**

- Cori's Haswell nodes are built from 2 Xeon processors (sockets)
  - Memory attached to each socket (fast)
  - Interconnect that allows remote memory access (slow == NUMA)
  - Improper memory allocation can result in more than a 2x performance penalty









RKELEY NATIONAL LABORATORY

# Modeling In-Core Performance Effects





# Data, Instruction, Thread-Level Parallelism...

Modern CPUs use several techniques to increase per core Flop/s

## **Fused Multiply Add**

- $w = x^*y + z$  is a common idiom in <u>kine</u> algebra
- CO se a ado (FMA)
- mutiply and add in a single pipeline so that it can complete FMA/cycle

## **Vector Instructions**

- Many HPC codes apply the same operation to a vector of elements
- Vendors provide vector instructions that apply the same operation to 2, 4, 8, 16 elements...

x [0:7] \*y [0:7] + z [0:7]

Vector FPUs complete 8 vector operations/cycle

## **Deep pipelines**

- is substantial.
- increase GHz
- FP bandwidth



The hardware for a FMA

Breaking a single FMA up into several smaller operations and pipelining them allows vendors to

Little's Law applies... need FP Latency \* independent instructions



# Data, Instruction, Thread-Level Parallelism...

- If every instruction were an ADD (instead of FMA), performance would drop by 2x on KNL or 4x on Haswell
- Similarly, if one had no vector instructions, performance would drop by another 8x on KNL and 4x on Haswell
- FP Divides can be even worse.
- Lack of threading will reduce performance by 64x on KNL.







# **Superscalar vs. instruction mix**

- Define in-core ceilings based on instruction mix...
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance







# Superscalar vs. instruction mix

- Define in-core ceilings based on instruction mix...
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance






# Superscalar vs. instruction mix

- Define in-core ceilings based on instruction mix...
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance









**BERKELEY LAB** 

NCE BERKELEY NATIONAL LABORATORY

# Modeling Cache Effects





 Naively, we can bound AI using only compulsory cache misses







- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower Al



 $AI = \frac{\#Flop's}{Compulsory Misses + Write Allocates}$ 



- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower Al
- Cache capacity misses can have a huge penalty



## $AI = \frac{\#Flop's}{Compulsory Misses + Write Allocates + Capacity Misses}$



42

- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower Al
- Cache capacity misses can have a huge penalty

Compulsory Misses + Write Allocates + Capacity Misses

Compute bound became memory bound

 $AI = \frac{\#Flop's}{\pi}$ 





**BERKELEY LAB** 

ENCE BERKELEY NATIONAL LABORATORY



Broadly speaking, there are three approaches to improving performance:





- Broadly speaking, there are three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)





- Broadly speaking, there are three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)





- Broadly speaking, there are three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)
- Minimize data movement (increase AI)







# Constructing a Roofline Model requires answering some questions...

ABORATORY



# **Questions can overwhelm users...**

What is my Properties of the target machine

## (Benchmarking)

FMA on my machine?

> What is my machine's DDR GB/s? L2 GB/s?

How much data did my kernel actually move? **Properties of an** application's execution **How many** flop (Instrumentation) do? How much did that divide hurt?



## **Fundamental** properties of the kernel constrained **by** hardware







**BERKELEY LAB** 

RKELEY NATIONAL LABORATORY

# To answer these questions, we need tools...





# **Node Characterization?**

- "Marketing Numbers" can be deceptive...
  - Pin BW vs. real bandwidth
  - TurboMode / Underclock for AVX
  - compiler failings on high-AI loops. ullet
- LBL developed the Empirical Roofline Toolkit (ERT)...
  - Characterize CPU/GPU systems lacksquare
  - Peak Flop rates
  - Bandwidths for each level of memory lacksquare
  - **MPI+OpenMP/CUDA == multiple GPUs**









# **Instrumentation with Performance Counters?**

- Characterizing applications with performance counters can be problematic...
  - **x** Flop Counters can be broken/missing in production processors
  - **x** Vectorization/Masking can complicate counting Flop's
  - x Counting Loads and Stores doesn't capture cache reuse while counting cache misses doesn't account for prefetchers.
  - **x** DRAM counters (Uncore PMU) might be accurate, but...
    - x are privileged and thus nominally inaccessible in user mode
    - may need vendor (e.g. Cray) and center (e.g. NERSC) approved Χ **OS/kernel** changes





# **Forced to Cobble Together Tools...**

- Use tools known/observed to work on NERSC's Cori (KNL, HSW)...
  - Used Intel SDE (Pin binary instrumentation + ٠ emulation) to create software Flop counters
  - Used Intel VTune performance tool (NERSC/Cray ٠ approved) to access uncore counters
- Accurate measurement of Flop's (HSW) and DRAM data movement (HSW and KNL)
- Used by NESAP (NERSC KNL application) readiness project) to characterize apps on Cori...

FOR USERS = Live Status = User Announcements = My HERBC	Home + For Users + Application Pe
User Announcements     My NERSC	ALCACIUMIC A
User Announcements     My NERSC	MEASURING A
My NERSC	MERCONING A
Getting Started	Arithmetic intensity is a measure
Connecting to NERSC	amount of memory accesses (B)
Accounts & Allocations	ratio (F/B). This application note
Computational Systems	Emulator Toolkit (SDE) and VT
Birrana & Ella Bustama	on using VTune can be found he
a Application Deduceron	Performance Model.
- Approaction Performance	
NESAP	Historically, processor manufact
Application Porting and	calculation. Some modern proce
Performance	provide counters for FLOPs. Ho
MPUS	memory accesses, and Vilune or
Performance and Debugging Tools	The SDE dynamic instruction tra
Measuring Arithmetic	instruction length, instruction on
Interneity	with SDE. In general the following
Data & Analytics	Edinor and Cod Phase 1
Job Logs & Statistics	surger and con Phase 1.
» Training & Tutorials	This application note provider a
a Bollowara	critical for real applications on bo
- Bulleton	more than a few minutes. And m
Policies	more than a rew minutes. And in
* User Burveys	An example command inc for P
NERSC Users Group	An example command line for S
= Help	
Staff Blogs	\$ srun -n 4 -c 6 sde -ivb -d -8
Request Repository Mailing	
List	s
	a Where
Manual Marine?	<ul> <li>Job is used to target Ed</li> </ul>
Need Help?	<ul> <li>Interview to target college</li> <li>Interview to target college</li> </ul>
Out-of-hours Status	<ul> <li>-a specines to only cone</li> </ul>
and Password help	<ul> <li>-iform 1 turns on compute</li> </ul>
Call operations:	<ul> <li>-omix specifies the outp</li> </ul>
1-800-06-NERSC, option 1	· Japanifes that each per
or 510-486-6821	<ul> <li> specnes mat each pri</li> </ul>
	<ul> <li>-global_region will inclu</li> </ul>
Account Support	
https://inim.nersc.gov	
accounts@nersc.gov	An example command line for V
1-800-66-NERSC, option 2	

### http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/









# **Initial Roofline Analysis of NESAP Codes**

**MFD**n **EMGeo PICSAR** 10000 10000 10000 Roofline Model **MSH** 1000 1000 1000 Roofline Model •wo/FMA GFLOP/s **GFLOP/s GFLOP/s** -wo/FMA Original 100 100 100 **SELL** 📔 1 RHS 10 10 10 🔺 4 RHS ♦ SB **2P** 8 RHS SELL+SB 1 nRHS+SELL+SB 0.1 0.1 0.1 0.01 1 10 1 10 Arithmetic Intensity (FLOP/byte) Arithmetic Intensity (FLOP/byte) 10000 10000 Roofline Model 1000 1000 Roofline Model -wo/FMA GFLOP/s **GFLOP/s** GFLOP Original wo/FMA **KNL** 100 100 **SELL** 📔 1 RHS 10 10 10 🔺 4 RHS 🔶 SB 8 RHS SELL+SB 1 1 1 nRHS+SELL+SB 0.1 0.01 0.1 0.1 10 1 10 1 10 1 Arithmetic Intensity (FLOP/byte) **Arithmetic Intensity (FLOP/byte) Arithmetic Intensity (FLOP/byte)** 







w/Tiling+Vect



# **Evaluation of LIKWID**

- LIKWID provides easy to use wrappers for measuring performance counters...
  - Works on NERSC production systems  $\checkmark$
  - Minimal overhead (<1%)  $\checkmark$
  - Scalable in distributed memory (MPI-friendly)  $\checkmark$
  - Fast, high-level characterization
  - No detailed timing breakdown or optimization advice X
  - Limited by quality of hardware performance counter X implementation (garbage in/garbage out)
- > Useful tool that complements other tools





# **Need an integrated solution...**

- Having to compose VTune, SDE, and plotting tools...
  - worked correctly and benefited NESAP's application readiness  $\checkmark$
  - forced users to learn/run multiple tools and manually parse/graph the output X
  - forced users to instrument routines of interest in their application X
  - lacked integration with compiler/debugger/disassembly X

### LIKWID was...

- fast and easy to use  $\checkmark$
- Suffered from the same limitations as VTune/SDE X
- ERT...
  - Characterized flops, and bandwidths (cache, DRAM)  $\checkmark$
  - Interoperable with MPI, OpenMP, and CUDA  $\checkmark$
  - Required users to manually parse/incorporate the output







# Intel Advisor

## Includes Roofline Automation...

- Automatically instruments applications (one dot per loop nest/function)
- Computes FLOPS and AI for each function (CARM)
- ✓ AVX-512 support that incorporates masks
- Integrated Cache Simulator<sup>1</sup> (hierarchical roofline / multiple Al's)
- Automatically benchmarks target system (calculates ceilings)
- Full integration with existing Advisor capabilities



### http://www.nersc.gov/users/training/events/roofline-training-1182017-1192017

<sup>1</sup>Technology Preview, not in official product roadmap so far.





BERKELEY LAB

# Hierarchical Roofline vs. Cache-Aware Roofline

...understanding different Roofline formulations in Advisor



# There are two Major Roofline Formulations:

- Hierarchical Roofline (original Roofline w/ DRAM, L3, L2, ...)...
  - Williams, et al, "Roofline: An Insightful Visual Performance Model for Multicore Architectures", CACM, 2009 •
  - **Chapter 4 of "Auto-tuning Performance on Multicore Computers"**, 2008 •
  - Defines multiple bandwidth ceilings and multiple Al's per kernel ٠
  - Performance bound is the minimum of flops and the memory intercepts (superposition of original, single-metric Rooflines) •

### **Cache-Aware Roofline**

- Ilic et al, "Cache-aware Roofline model: Upgrading the loft", IEEE Computer Architecture Letters, 2014 •
- Defines multiple bandwidth ceilings, but uses a single AI (flop:L1 bytes) •
- As one looses cache locality (capacity, conflict, ...) performance falls from one BW ceiling to a lower one at constant AI •
- Why Does this matter?
  - Some tools use the Hierarchical Roofline, some use cache-aware == Users need to understand the differences •
  - Cache-Aware Roofline model was integrated into production Intel Advisor •
  - Evaluation version of Hierarchical Roofline<sup>1</sup> (cache simulator) has also been integrated into Intel Advisor •





<sup>&</sup>lt;sup>1</sup>Technology Preview, not in official product roadmap so far.

# **Hierarchical Roofline**

- Captures cache effects
- AI is Flop:Bytes after being *filtered by* lower cache levels
- Multiple Arithmetic Intensities (one per level of memory)
- Al *dependent* on problem size (capacity misses reduce AI)
- Memory/Cache/Locality effects are observed as decreased AI
- Requires *performance counters or cache simulator* to correctly measure Al

## **Cache-Aware Roofline**

- Captures cache effects
- Al is Flop:Bytes as presented to the L1 cache (plus non-temporal stores)
- Single Arithmetic Intensity
- Al *independent* of problem size
- Memory/Cache/Locality effects are observed as decreased performance
- Requires static analysis or *binary instrumentation* to measure Al





# **Example: STREAM**

### • L1 Al...

- 2 flops
- 2 x 8B load (old)
- 1 x 8B store (new)
- = 0.08 flops per byte

### No cache reuse…

• Iteration i doesn't touch any data associated with iteration i+delta for any delta.

### ... leads to a DRAM AI equal to the L1 AI

#pragma omp parallel for
for(i=0;i<N;i++){
 Z[i] = X[i] + alpha\*Y[i];
}</pre>







# **Example: STREAM**

## **Cache-Aware Roofline**



# Example: 7-point Stencil (Small Problem)

### L1 Al...

- 7 flops ٠
- 7 x 8B load (old) ٠
- 1 x 8B store (new) ٠
- = 0.11 flops per byte ٠
- some compilers may do register shuffles to reduce the • number of loads.

### Moderate cache reuse...

- old[ijk] is reused on subsequent iterations of i,j,k ٠
- old[ijk-1] is reused on subsequent iterations of i. ٠
- old[ijk-jStride] is reused on subsequent iterations of j. ٠
- old[ijk-kStride] is reused on subsequent iterations of k. •

```
... leads to DRAM AI larger than
the L1 AI
```

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){</pre>
for(j=1;j<dim+1;j++){</pre>
for(i=1;i<dim+1;i++){</pre>
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk
                 + old[ijk-1
                 + old[ijk+1
                 + old[ijk-jStride]
                 + old[ijk+jStride]
                 + old[ijk-kStride]
                 + old[ijk+kStride];
}}}
```





## **Example: 7-point Stencil (Small Problem) Hierarchical Roofline Cache-Aware Roofline**





## **Example: 7-point Stencil (Small Problem) Hierarchical Roofline Cache-Aware Roofline**



Peak Flop/s



## Example: 7-point Stencil (Large Problem) **Hierarchical Roofline Cache-Aware Roofline**



# Peak Flop/s



## Example: 7-point Stencil (Observed Perf.) **Hierarchical Roofline Cache-Aware Roofline**





# Peak Flop/s



## Example: 7-point Stencil (Observed Perf.) **Hierarchical Roofline Cache-Aware Roofline**





# Peak Flop/s





**BERKELEY LAB** 

RKELEY NATIONAL LABORATORY

# Example of Roofline in Practice





# **Sparse Matrix Vector Multiplication**

- What's a Sparse Matrix ?
  - Most entries are 0.0 •
  - Performance advantage in only storing/operating on the nonzeros ٠
  - Requires significant meta data to reconstruct the matrix structure
- What's SpMV ?
  - Evaluate y=Ax where A is a sparse matrix, x & y are dense vectors •
- Challenges
  - Very low arithmetic intensity (often <0.166 flops/byte)
  - Difficult to exploit ILP (bad for pipelined or superscalar), •
  - Difficult to exploit DLP (bad for SIMD) •







# **Roofline model for SpMV**

- Double precision roofline models
- In-core optimizations 1..i
- DRAM optimizations 1..j

 $\label{eq:GFlops} \textbf{GFlops}_{i,j} \textbf{(AI)} = min \begin{cases} \textbf{InCoreGFlops}_i \\ \textbf{StreamBW}_j & \textbf{AI} \end{cases}$ 

• FMA is inherent in SpMV (place at bottom)





# **Roofline model for SpMV**

(overlay arithmetic intensity)

- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166</li>




## **Roofline model for SpMV**

(out-of-the-box parallel)

- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166</li>
- For simplicity: dense matrix in sparse format





### **Roofline model for SpMV**

(NUMA & SW prefetch)

- compulsory flop:byte ~ 0.166
- utilize all memory channels





### **Roofline model for SpMV**

#### (matrix compression)

- Inherent FMA
- Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions





## **Roofline on CPUs and GPUs**

- In 2010, we began to use Roofline to compare CPU and GPU performance for a variety of double-precision kernels
  - Flop/s were theoretical book values;
  - Memory bandwidth from STREAM or SHOC
  - AI was based on compulsory data movement ۲
  - Optimized kernel performance ۲ was well-correlated with Roofline for both platforms.
  - Some irregular applications (PIC) underperformed and motivated Further study.







# Questions?







# Backup







# nte Acvisor: Introduction and General Usage

\*DRAM Roofline and OS/X Advisor GUI: These are preview features that may or may not be included in mainline product releases. They may not be stable as they are prototypes incorporating very new functionality. Intel provides preview features in order to collect user feedback and plans further development and productization steps based on the feedback.





### Intel Advisor

- Integrated Performance Analysis Tool
  - Performance information including timings, flops, and trip counts
  - Vectorization Tips
  - Memory footprint analysis
  - Originally used the Cache-Aware Roofline Model
  - All connected back to source code

### CRD/NERSC began a collaboration with Intel

- Ensure Advisor runs on Cori in user-mode
- Push for Hierarchical (Integrated) Roofline
- Make it functional/scalable to many MPI processes across multiple nodes

80

• Validate results on NESAP, SciDAC, and ECP codes

NESAP is NERSC's KNL application readiness project SciDAC is the DOE Office of Science's Scientific Discovery thru Advanced Computing program ECP is the DOE's Exascale Computing Project



## Intel Advisor (Useful Links)

#### Background

- https://software.intel.com/en-us/intel-advisor-xe
- https://software.intel.com/en-us/articles/getting-started-withintel-advisor-roofline-feature
- https://www.youtube.com/watch?v=h2QEM1HpFgg

#### **Running Advisor on NERSC Systems**

http://www.nersc.gov/users/software/performance-anddebugging-tools/advisor/

	2 - 0	software.intel.com	0	
Vectorization and 1	Threading are Cr	ucial to Performanc	e Overview	
On modern ersterner, it is here	aning an stal in both up do	dae from ALON or COMPA lands with		
On modern processors, it is beo and thread software to realize th	e full performance potential	tof the processor. In some cases	ons) Details	
that is vectorized and threaded of	an be up to 187X faster the	an unthreaded/unvectorized code	-and Try & Bu	
about 7X faster than code that is new processor generation.	only threaded or vectorize	<ol> <li>And that gap is growing with er</li> </ol>	FAO	
and branches Brunners				
200,000 St. The I	Difference is Growing Wit	th 💦 💿 🖉		
Each M	lew Generation of Hardw	are		
is Berl		197		
inter 60 mm		107		C
				P
4			HOME ABOUT SCIENCE AT N	ERSC SYSTEMS
2007 2009	2010 2012 201	2014 2016		Home - For Use
Processor: X5472 X5570 Indenamed Hasteristen, Nethalian	X5680 E5-260 E5-260 Westmans Sandy Bridge Ivy Br	13 2014 2019 10 12 E5-2500 v3 E5-2500 100 Illinovill Broadie	FOR USERS	Home - For Us
Software and workloads used in performa	nce tests may have been optimized t	for performance only on troot es	Live Status	ΔΠΛΙΟ
operations and functions. Any change to a and performance tests to assist you in full	ny of those factors may cause the re y evaluating your contemplated pury	suits to vary. You should compare haues, including the performant	User Announcements     My NERSC	
when combined with other products. For a	sore information go to http://www.in	tel.com/performance.	Getting Started	Introduct
Threaded plus vectorized can be new hardware generation. Detail	much faster than either or	te alone. The gap is gro	Connecting to NERSC     Accounts & Allocations	Intel Advices
	-		Computational Bystems	applications of
an Departure 1720 Contrained Contra		and points	Btorage & File Bystems	
Plannay Shruphant B Released by	and the set of the set		Application Performance     Data & Analytics	Vectoriza     specifies
(K) III Aunction Call Stees and Longes 🛔 Yester Source	Salt Time= Type	PLOPS AV Vectorization* Vector	- Job Logs & Statistics	data reorg
In the second se	LTD LTD Technical	8.1011 8.1019 8 sectors. 2011 8.1081 8.005 8 sectors.	Training & Tutorials     Software	Threading
Step in (1) if topplite     Sep in (1) topplite     Sep in (1) topplite     Sep in (1) topplite	- 0.07% Erel Autor State	E color de	User Environment	design, ti code dev
H C Doog in 1051 at hospell(A2001) C #19au H C Doog in 1051, hompigaeaded, Ar., M #19au	A.716     Vectorized (     Antice Vectorized (     Antice Vectorized (	1.714 0.020 eVA	Using Shifter and Docker	
4 Indexed and the second secon	format the party mapping	ania hadron unu in mat a	Applications	For more infor
in implementation. Implement or	ly the options that have a l	high return on investmer	Compilers	
Data Dalaan Mastaalaati	o Ontinination and	Threadles Deale	Version Control Tools	Using Int
Data-Driven vectorizatio	on Optimization and	Threading Desig	Programming Libraries	To launch Adv
You need good data to make go	od design decisions. What	loops should be threade	Performance and Debugging Tools	results using t
vectorized first? Is the performant larger core counts? Does this lo	too gain worth the effort? W to have a dependency that	vil the threading perform prevents vectorization?	DOT Tetebine	Compilio
trip counts and memory access	patterns? Have I vectorized	efficiently with the later	008	Additional
I using older SIMD instructions?			STAT and ATP	In order to cor
Vectorization Optimizat	ion: Guidance to Spe	eed up your Appli	OCOB and lights Valgrind	Cray Compile
			PM	When using the
		Look for us	CreyPat MAP	Advisor. To co
			Viline	
			Darshan	cc -g -dy
			Inspector	Here, the -g o
			Intel Trace Analyzer and Collector	dynamically lit
			Grid Software and Services	staticary by 6
			Pulcies     User Surveys	Without the -d
			NERSC Users Group	S. module
			- Help	% cc -g -





### **Using Intel Advisor at NERSC**

#### Compile...

use '-g' when compiling

### Submit Job...

% salloc -perf=vtune <<< interactive sessions; --perf only needed for DRAM Roofline</pre>

-or-

**#SBATCH** -perf=vtune

<<< batch submissions: --perf only needed for DRAM Roofline</pre>

#### Benchmark...

% module load advisor

% export ADVIXE\_EXPERIMENTAL=roofline\_ex <<< only needed for DRAM Roofline

% srun [args] advixe-cl -collect survey -no-stack-stitching -project-dir \$DIR -- ./a.out [args] % srun [args] advixe-cl -collect tripcounts -flops-and-masks -callstack-flops -project-dir \$DIR -- ./a.out [args]

#### Use Advisor GUI...

% module load advisor

% export ADVIXE\_EXPERIMENTAL=roofline\_ex <<< only needed for DRAM Roofline</pre>

% advixe-qui \$DIR













/elcome e000 (read-only) 🗶							
🗖 Elapsed time: 50.50s 🏼 😽 🧕 Vec	torized 🛯 Not Vectorized 🍯				OFF Smart M	lode <sup>®</sup>	Q
ILTER: All Modules - All Sources	✓ Loops And Functions ✓ A	ll Threads 👻				INTEL	ADVISOR 201
Summary 🛠 Survey & Boofline 🦸	Refinement Reports						
Summary Call Cites and	r Reinferie Reports			1 1	FLOPS		¥.
E = Function Call Sites and Loops	♦ Vector Issues	Self Time Total	Time Type		GELOPS		A1
■©[loop in bench_stencil_ver4\$		159 595s⊡ 159 <sup>g</sup>	95s Vectoriz	ed (Body)	23.083		0.117
	Ineffective peeled/	. 159.953s <b>□</b> 159.9	53s Vectoriz	ed (Body;	Re 16.274		0.117
[loop in bench stencil ver2\$	I Ineffective peeled/	. 160.035s 160.0	35s Vectoriz	ed (Body;	Peel 15.662		0.117
[loop in bench_stencil_ver1\$	I Ineffective peeled/	. 159.307s 159.3	07s Vectoriz	ed (Body;	Peel 10.218		0.117
🗵 🖉 [loop in bench_stencil_ver0\$	🔲 🛛 🕯 1 Potential underutil	. 157.994s <b>=</b> 157.9	94s🔲 Scalar		9.009		0.117
Source Top Down Code Analytics	Assembly 💡 Recommendati	ions 📮 Why No Ve	ctorization?				
ile: stencil_v2.c:108 bench_stenc	cil_ver2\$omp\$parallel_for@	102	· · ·				
in.	Source		Total Time	% Lo	oop/Function Time	%	Traits
JS <sup>1</sup> IOT(LILE=U; LILE <jiiles*< td=""><td>κιιιes;ιιιe++){</td><td></td><td>0.0105</td><td></td><td>· · · · · · · · · · · · · · · · · · ·</td><td></td><td></td></jiiles*<>	κιιιes;ιιιe++){		0.0105		· · · · · · · · · · · · · · · · · · ·		
$1nt kLo = 16^{+}(t1te/)T$	iles);						
)5 intilo = 16*(tilo%iT	+){		0.0085				
)5 int jLo = 16*(tile%jT )6 ⊞ for(k=kLo:k <klo+16:k+< td=""><td>+){</td><td></td><td>0.092s</td><td>[</td><td></td><td></td><td></td></klo+16:k+<>	+){		0.092s	[			
D5 int jLo = 16*(tile%jT D6    for(k=kLo;k <klo+16;k+ D7    for(j=jLo;j<jlo+16;j+< td=""><td></td><td></td><td>1.500s</td><td></td><td>160.035s</td><td></td><td></td></jlo+16;j+<></klo+16;k+ 			1.500s		160.035s		
05 int jLo = 16*(tile%jT 06							
05 int jLo = 16*(tile%jT 06   for(k=kLo;k <klo+16;k+ 07   for(j=jLo;j<jlo+16;j+ 08   for(i=0;i<dim;i++){ 09 int ijk = i + j*jSt</dim;i++){ </jlo+16;j+ </klo+16;k+ 	ride + k*kStride;						FMA
<pre>05 int jLo = 16*(tile%jT 06    for(k=kLo;k<klo+16;k+ 07    for(j=jLo;j<jlo+16;j+ 08    for(i=0;i<dim;i++){ 09 int ijk = i + j*jSt 10 new[ijk] = -6.0*old</dim;i++){ </jlo+16;j+ </klo+16;k+ </pre>	ride + k*kStride; [[ijk ]		26.216s				
05 int jLo = 16*(tile%jT 06   for(k=kLo;k <klo+16;k+ 07   for(j=jLo;j<jlo+16;j+ 08   for(i=0;i<dim;i++){ 09 int ijk = i + j*jSt 10 new[ijk] = -6.0*old</dim;i++){ </jlo+16;j+ </klo+16;k+ 	ride + k*kStride; [[ijk ] Se	elected (Total Time	26.216s ): 1.500s				
<pre>05</pre>	ride + k*kStride; [[ijk ] Se	elected (Total Time	26.216s ): 1.500s				



		No	Machine - NER	SC				
/global/cscratch1/sd/tkoskela/dram	m_roofline/st	encil/advi.stencil.a	ug2.16 - In	tel Advisor <	@cori0	5> <2>		
File View Help								
Welcome e000 (read-only) 🗶								
📾 Elapsed time: 50.50s 🤻 🗖 Ve	ctorized 🛯 🖉 N	lot Vectorized 🍯					OFF	S
FILTER: All Modules - All Sources	✓ Loops An	d Functions 👻 A	ll Threads	<b>-</b>				
Summary Survey & Roofline	Refinement	t Reports		_				
Function Coll Sites and	-							FLOPS
$\stackrel{\text{Function Call Sites and}}{\text{Loops}}$	♦ 💡 Vecto	or Issues	Self Time	Total Time	Туре			GELOF
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □			159.595s	159.595s	Vecto	rized (Bodv)		23.08
E ⊡op in bench_stencil_ver3	🔲 🔋 1 Ine	ffective peeled/	159.953s	159.953s	Vecto	rized (Body;	Re	16.27
	🔲 💡1 Ine	ffective peeled/	160.035s	160.035s	Vecto	rized (Body;	Peel	15.66
⊡op in bench_stencil_ver1	🔲 💡1 Ine	ffective peeled/	159.307s	159.307s	Vecto	rized (Body;	Peel	10.21
Icop in bench_stencil_v	📃 🔋 1 Po	tential under	157.994s 🛛	157.994s	Scala	r		9.00
				1	1	-		
Source Top Down Code Analytics	Assembly	💡 Recommendati	ons 📮 Wh	y No Vectoriza	ation?			
Function Call Sites and Loops			Total Tim	e % Total T	ime	Self Time	Туре	
□_INTERNAL_26src_z_Lir	nux_util_cpp_	2d702c13::[Open	M 93.8%	755.84	3s	0.000s	Funct	ion
⊡©[loop in _INTERNAL_26	src_z_ IN	TERNAL 26	src z Linu	x util cpp 2d	702c1	3··[OpenMP	worke	rl
□kmp_launch_thread			93.8%	735.84	-35	0.0005	Funct	ion
⊟ő[loop inkmp_launch_	thread at km	p_runtime.cpp:56	5. 93.8%	755.84	3s	0.000s	Scalar	r
⊟[OpenMP dispatcher]			93.3%	752.00	)0s	0.000s	Funct	ion
⊞bench_stencil_ver2\$o	mp\$parallel_	for@102	18.7%	150.90	)3s	0.120s	Funct	ion
⊞bench_stencil_ver3\$o	mp\$parallel_	for@146	18.7%	150.47	1s	0.000s	Funct	ion
⊞bench_stencil_ver4\$o	mp\$parallel_	for@193	18.6%	149.98	39s	0.000s	Funct	ion
Bonch stopsil vorl¢o	mn¢nərəlləl	far@62	10 60/	140.74	30	0.000cl	Eunct	ion
📕 🔜 Madvixe-gui		₃ 🔄 emacs-gtk(	cori05-bor	nd0.224		🔳 cori :		

	$\odot$	×
		•
mart Mode	° Q L Advisor 201	7
PS ▼ 3 4 2 3 9	AI 0.117 0.117 0.117 0.117 0.117	
	FLOPS	
	0.998 0.998 0.998 0.998	
	1.003 1.083 1.097 1.540	
	0.606	



Image: A state in the state		toncil/odvi otoncil	NoMachine - NER	SC	a coriot	2	
File View Help	im_roonne/s	tench/advi.stench	.aug2.10 - In		@conos>		
Welcome e000 (read-only) 🗶							
🛛 📾 Elapsed time: 50.50s 😽 🗿 Ve	ectorized 🚨	Not Vectorized 🦉				OFF	S
FILTER: All Modules - All Sources	- Loops A	nd Functions 👻	All Threads	-			
😤 Summary 🛭 🕊 Survey & Roofline	② Refineme	nt Reports					
The Function Call Sites and			0. KT		-		FLOPS
8 ± E Loops	♦ Vect	or Issues	SelfTime	lotal lime	Туре		GFLOP
🚊 🗵 [loop in bench_stencil_ver4			159.595s	159.595s	Vectoriz	ed (Body)	23.083
a and a stencil_ver3	🔲 🕯 1 In	effective peeled/	159.953s	159.953s	Vectoriz	ed (Body; Re	16.274
∃op in bench_stencil_ver2	🔲 🖓 1 Inc	effective peeled/	160.035s	160.035s	Vectoriz	ed (Body; Peel	15.662
∎º [loop in bench_stencil_ver1	📃 💡 1 Inc	effective peeled/	. 159.307s	159.307s	Vectoriz	ed (Body; Peel	10.218
Iloop in bench_stencil_v	🗌 🔋 1 Pc	otential under	157.994s	157.994s 🗖	Scalar		9.00
Source Top Down Code Analytics	s Assembly	💡 Recommenda	tions 📮 Why	y No Vectoriza	ation?		
Loop in bench_stencil_ver0\$omp\$para	allel_for	Average T	rip Count	ts: 512		GFLOPS: 9	.0087
at stencil_v2.c:29	_	Instruction	Mix <sup>∞</sup>		⊙ (	Code Optim	izatio
C 157. Scalar Total tim	994s	Memory: <b>5</b> <b>4</b> Other: <b>4</b> Registers: <b>9</b>	Compute: 9 Number d	Mixed <sup>®</sup> : of Vector	C C Ir	Compiler: Intel( Compiler for ap ntel(R) 64,	R) C In plicatic
157.99 Self time	)4s				V	/ersion: 17.0.2.	174 Bi
🚜 🔚 🔛 Advixe-gui		🔒 😔 emacs-gtk	@cori05-bor	nd0.224		cori :	

	© (\$)	×
		•
mart Mode	° 0,	
INTE	L ADVISOR 201	7
	<i>a</i>	П
°S▼	AI	
3	0.117	11
4	0.117	
2	0.117	
8	0.117	
9 📖	0.117	
73	$\bigcirc$	L
13	Ŭ	Ш
		Ш
ons	۲	
tol(D) 64		
ons running	y on	
	010	
uild 20170	213	



					NoMachine - NER	SC				
🚰 💽 /gl	lobal/cscratch1/s	sd/tkoskela/dra	am_roofline/s	tencil/advi.stenci	l.aug2.16 - In	tel Advisor <(	@cori05>	<2>		
File Vi	iew Help									
Welco	me e000 (read	-only) 🗶								
in E	Elapsed time: 50	.50s 🤻 🖸 Ve	ectorized 👩	Not Vectorized 🧖	]				OFF	S
FILTER	R: All Modules	<ul> <li>All Sources</li> </ul>	- Loops A	nd Functions 👻	All Threads	-				
🤗 Sur	mmary 🛠 Surv	ev & Boofline	A Refineme	ent Reports						
A Sul				ine nepores						FLODS
RO 🗉	E Loops	l Sites and	♦ 😵 Vect	tor Issues	Self Time	Total Time	Туре		-	CELOP
		stonsil vor4			150 505c	150 505c	Voctoriz	ad (Radu)		32.092
z	[loop in bench_	stencil_ver4		offective peoled/	159.5958	159.5955	Vectorize	ed (Body)	20	16.27/
	[loop in bench_	stencil ver2		effective peeled/	160.035c	160.035s	Vectorize	ed (Body: P	Ne	15 662
	[loop in bench_	stencil ver1		effective peeled/	150.0335	150.033s	Vectorize	ed (Body: P	Pool	10.219
	[loop in bench	stencil v		effective peeled/	157 994cl	157 994s	Scalar	eu (Bouy, F	eer	9 00
		v		stentiar under	157.5543	137.3343	Scalar			5.00
Sourc	ce Top Down	Code Analytics	s Assembly	🖉 💡 Recommenda	tions 📮 Wh	y No Vectoriza	ntion?			
Modu	ile: a.out!0x40	1690				•				
	Address Lin			Assembly			Т	otal Time	%	Self
body	0x401690	Block 1:		Assembly					/0	Jen
	0x401690 34	vmovsdq (%	rbp,%rdx,8	), %xmm1				0.996s	[	
	0x401696 31	leal (%r13	,%r12,1), <sup>9</sup>	%r11d				1.728s	[	
	0x40169b 31	movsxd %r11	d, %r11					1.008s	[	
	0x40169e 29	inc %r10d						1.794s	(	
	0x4016a1 36	vmovsdq (%	rbp,%rdi,8	), %xmm2				0.944s	[	
	0x4016a7 29	add %r8, %r	dx					49.773s	1	4
	0x4016aa 29	add %r8, %r	di					1.456s		
					Se	lected (Total T	ime):	0s		
				: Compare at	CoriOE hor	d0 224		cori		
	Auvixe-qu	1		3 emacs-gti	(@C01103-b01	100.224		con :	_	_

			<ul><li></li></ul>	×
				•
mart Mo	ode	9	C	2,
	INTE	LAD	VISOR 20	117
				1
°S▼		AI		
3		0.1	117	
4	]	0.3	117	
2	0.1	-11		
B		0.1	117	41
9	_	0.	117	
Time	0/		Traite	10
Time	%		Traits	-11
0.996s[				
1.728s (				
1.008s (				
1.794s				
0.944s[				
9.773s1				
1.4565				







## Intel Advisor: Stencil Roofline Demo\*

\*DRAM Roofline and OS/X Advisor GUI: These are preview features that may or may not be included in mainline product releases. They may not be stable as they are prototypes incorporating very new functionality. Intel provides preview features in order to collect user feedback and plans further development and productization steps based on the feedback.



## 7-point, Constant-Coefficient Stencil

- Apply to a 512<sup>3</sup> domain on a single NUMA node (single HSW socket)
- Create 5 code variants to highlight effects (as seen in advisor)

ver0.	Baseline: thread over outer loop (k), but prevent vectorization								
	#pragma novector	// prevent simd							
	<pre>int ijk = i*iStride + j*jStride + k*kStride;</pre>	<pre>// variable iStride to confuse</pre>							
ver1.	Enable vectorization								
	int ijk = i + j*jStride + k*kStride;	// unit-stride inner loop							
ver2.	Eliminate capacity misses								
	2D tiling of j-k iteration space	// working set had been O(6MB) per							
ver3.	Improve vectorization								
	Provide aligned pointers and strides								
ver4.	Force vectorization / cache bypass								
	<pre>assume(jstride%8 == 0);</pre>	<pre>// stride by variable is still</pre>							
	<pre>#pragma omp simd, vector nontemportal</pre>	<pre>// force simd; force cache byp</pre>							



e the compiler

<sup>.</sup> thread

aligned pass



Alabal/accentabl/ad/thealestale/deems wasfling/atomail/aduit atomail awar2.16 Intel Advise					
global/cscratch1/sd/tkoskela/dram_roofline/stencil/advi.stencil.aug2.16 - Intel Adviso	or <@cori05> <2>			$\odot$	×
le View Help					
/elcome e000 (read-only) 🗶					•
🖬 Elapsed time: 50.50s 😽 🖸 Vectorized 💿 Not Vectorized 🔨		OFF	Smart Mod	le°	Q,
ILTER: All Modules - All Sources - Loops And Functions - All Threads -				<b>TEL ADVISOR</b>	2017
Summary Survey & Roofline & Refinement Reports					
	8		Lacens	1	
	A1	W Vectorized	Loops	Colin Fo	
GFLOPS	AI	vector I	Efficiency	Gain Es	. V
Image: Second Stepsile Ste	0.117	AVX2	80%	2.27X	4
= 1159.9555 = 15.4.	0.117	AVX2	80%	] 3.33X	4
$\mathbb{P}^{[00p]}$ in bench stencil ver1\$0 $\mathbb{P}^{1100.0333}$ 10.V. 15.002	0.117	AVX2	80%	3.21x	4
Image: Second steps:         Image: Se	0.117		-	J.21X	-+
		-		-	
Source Top Down Code Analytics Assembly 💡 Recommendations 🖻 Why No Vector	orization?				
ile: stencil v2 c·29 bench stencil ver0\$omn\$parallel for@25					
net stenen_vztetzs benen_stenen_veroponippparanet_tot@zs					
in. Source	Total Time %	Loop/Functio	on Time	% Trait	s
in. Source 5 #pragma omp parallel for	Total Time % 9.890s	Loop/Functio	on Time	% Trait	s
<pre>incloseded_vzecizo_bened_stelled_verosponpopulatel_for@zo in. Source 5 #pragma omp parallel for 6   for(k=1;k<dim+1;k++){< pre=""></dim+1;k++){<></pre>	Total Time % 9.890s	Loop/Function	on Time	% Trait	s
<pre>incloseden_viewsbenen_stellen_verosompopulatiel_for@iss in. Source 5 #pragma omp parallel for 6  mu for(k=1;k<dim+1;k++){ 7  mu for(j=1;j<dim+1;j++){< pre=""></dim+1;j++){<></dim+1;k++){ </pre>	Total Time % 9.890s	Loop/Function	on Time	% Trait	s
<pre>incluster_references bench_steric_references population_references in. Source 5 #pragma omp parallel for 6   for(k=1;k<dim+1;k++){ #pragma="" 7="" 8="" <="" for(j="1;j&lt;dim+1;j++){" novector="" pre=""></dim+1;k++){></pre>	Total Time % 9.890s	Loop/Function	on Time	% Trait	5
<pre>incluster_vertex_bench_steller_vertex_binpsparamet_for@25 in. Source 5 #pragma omp parallel for 6   for(k=1;k<dim+1;k++){ #pragma="" 7="" 8="" 9="" <="" for(i="1;i&lt;dim+1;i++){" for(j="1;j&lt;dim+1;j++){" novector="" pre=""></dim+1;k++){></pre>	Total Time         %           9.890s         102.403s	Loop/Function	on Time	% Trait	5
<pre>incluster_order_or</pre>	Total Time       %         9.890s       102.403s	Loop/Function	on Time	% Trait	ts
<pre>include definition of the second paper and th</pre>	Total Time       %         9.890s          102.403s          53.651s	Loop/Function	on Time	% Trait	ts
<pre>in. Source 5 #pragma omp parallel for 6  for(k=1;k<dim+1;k++){ #pragma="" +="" 0="" 1="" 2<="" 7="" 8="" 9="" ]="" for(i="1;i&lt;dim+1;i++){" for(j="1;j&lt;dim+1;j++){" ijk="i*iStride" int="" j*jstride="" k*kstride;="" new[ijk]="-6.0*old[ijk" novector="" td=""><td>Total Time       %         9.890s          102.403s          53.651s          102.403s</td><td>Loop/Function</td><td>on Time</td><td>% Trait</td><td>ts  </td></dim+1;k++){></pre>	Total Time       %         9.890s          102.403s          53.651s          102.403s	Loop/Function	on Time	% Trait	ts
<pre>in. Source 5 #pragma omp parallel for 6  for(k=1;k<dim+1;k++){ 7  for(j=1;j<dim+1;j++){ 8 #pragma novector 9  for(i=1;i<dim+1;i++){ 0  int ijk = i*iStride + j*jStride + k*kStride; 1  new[ijk] = -6.0*old[ijk ] 2</dim+1;i++){ </dim+1;j++){ </dim+1;k++){ </pre>	Total Time       %         9.890s       -         102.403s       -         53.651s       -         102.403s       -	Loop/Function	on Time	% Trait	
<pre>in. Source 5 #pragma omp parallel for 6   for(k=1;k<dim+1;k++){ #pragma="" +="" 0="" 1="" 2="" 7="" 8="" 9="" ]="" for(i="1;i&lt;dim+1;i++){" for(j="1;j&lt;dim+1;j++){" ijk="i*iStride" int="" j*jstride="" k*kstride;="" new[ijk]="-6.0*old[ijk" novector="" old[ijk-istride]<="" td=""><td>Total Time       %         9.890s      </td><td>Loop/Function</td><td>on Time</td><td>% Trait</td><td>S</td></dim+1;k++){></pre>	Total Time       %         9.890s	Loop/Function	on Time	% Trait	S























					NoMachi	ne - NERSC			_
Image: Second secon	scratch1	/sd/tk	C	RAM	Ro	ofline*			
Welcome	× 000								
D Elapse	l time: 50	).40s 🄀 👩 Veci	torized 🖸 N	ot Vectorized	5 MKL				OFF Sma
FILTER: All	Modules	✓ All Sources	<ul> <li>Loops An</li> </ul>	d Functions 👻	All Th	eads 👻			
Summar	/ 🏷 Surv	/ey & Roofline	Refinemen	t Reports					
Perfor	mance (G	FLOPS)	<b>k Q</b> (10)	<ul> <li>★ → × □</li> </ul>	•   =	Use Single-Thr	eaded Roo	fs 🛛 🛛 🕞	Show Hierarchica
	L1 Band	Width: 1.1e+4 GB/ Width: 3542.01 GE	Sec Vsec						DP Vector Add Pea
i i	L3Band	andwidth: 128.58	GB/sec?	@ <del>-</del>		0			
0.6	B-			Iloop in Total P Total L	<b>n bench</b> erformai 1 Arithm	_stencil_ver0\$o nce: 5.45 GFLOP etic Intensity: 0.1	<b>mp\$paralle</b> S 7 FLOP/Byte	l_for@2	5 at stencil_v2.c:26
		0.05		Self Ela	apsed Ti	me: <b>0.000</b> s	,		
			<b>T F</b>	Total F	Inneed T	ime 10 000 c			
	Self Elaps	ed Time: 0.000 s	I otal Elap	sed I im	lapseu i	IIIIe. 10.000 S			
Source Te	op Down	ed Time: 0.000 s Code Analytics	Assembly	Recommend	lations	Why No Vecto	orization?		
Source To File: stend	Self Elaps op Down il_v2.c:2	ed Time: <b>0.000</b> s Code Analytics 6 bench_stencil	Assembly	Recomment \$parallel_for(	lations	Why No Vecto	orization?		
Source To File: stend Lin.	Self Elaps op Down il_v2.c:2	ed Time: <b>0.000</b> s Code Analytics 6 bench_stencil	Assembly ver0\$omp Source	<pre>sed lim_fotalL</pre>	lations @25	Why No Vecto	orization?	2 %	Loop/Function Tir
Source To File: stend Lin. 24 <sup>th</sup> whi 25 #	Self Elaps op Down il_v2.c:2 le(Elaps pragma o	ed Time: 0.000 s Code Analytics 6 bench_stencil edTime < TIME) mp parallel for	Assembly ver0\$omp Source {	<pre>sed lim_fotalL</pre>	lations @25	Why No Vecto	orization? Total Time	e %	Loop/Function Tin
Source To File: stend Lin. 24 <sup>th</sup> whi 25 # 26 <sup>th</sup> f	Self Elaps op Down il_v2.c:2 le(Elaps pragma o or(k=1;k	<pre>sed Time: 0.000 s Code Analytics 6 bench_stencil edTime &lt; TIME) mp parallel for <dim+1;k++){< pre=""></dim+1;k++){<></pre>	I otal Elap: Assembly ver0\$omp Source { r	<pre>sed lim_fotalL</pre>	lations @25	Why No Vecto	orization? Total Time	e %	Loop/Function Tir
Source To File: stend Lin. 24 <sup>w</sup> whi 25 <i>#</i> 26 <sup>w</sup> f 27 <sup>w</sup> f	Self Elaps op Down il_v2.c:2 le(Elaps pragma o or(k=1;k or(j=1;j	<pre>sed Time: 0.000 s Code Analytics 6 bench_stencil edTime &lt; TIME) mp parallel fo <dim+1;k++){ <dim+1;j++){<="" pre=""></dim+1;k++){></pre>	I otal Elap Assembly _ver0\$omp Source { r	<pre>sed lim_fotariz   Recomment   \$parallel_for(</pre>	lations @25 Selecto	Why No Vector	Total Time	e %	Loop/Function Tin





•••					NoMach	ne - NERSC			_
Image: Second secon	elp	sd/tk	C	DRAM	Ro	ofline*			
Welcome e0	00 X								
Elapsed	time: 50	.40s 🄀 🖸 Vec	torized 👩 N	lot Vectorized	MKI				OFF Sma
FILTER: All N	Iodules	All Sources	<ul> <li>Loops An</li> </ul>	d Functions 👻	All Th	reads 🔻			
Summary	🍫 Surv	ey & Roofline	Refinemen	t Reports					
Perform	ance (G	FLOPS)	R Q 🖑	◆ → × 🗊	-   =	Use Single-Thr	eaded Roo	fs 🛛   6	Show Hierarchic
1688.18	L1 Band	width: 1.1e+4 GB/ width: 3542.01 GE	sec 3/sec 3/sec						DP Vector Add Pea
	L3 Band	andwidth: 128.58	GB/sec?	@	Ø	0	-		
0.00	Div			[loop in Total Period	i <b>bench</b> erforma	_stencil_ver1\$on nce: 10.18 GFLO	m <b>p\$paralle</b> l PS	_for@6	2 at stencil_v2.c:63
0.66	1			Total L1	Arithm	etic Intensity: 0.1	7 FLOP/Byte	;	
		0.05		Self Ela	psed Ti ansed T	me: <b>0.000</b> s Time: <b>9 972</b> s			
S	elf Elaps	ed Time: <b>0.000</b> s	Total Elap	sed Tim	apseu	ine. <b>3.372</b> 5			
Source Top	Down	Code Analytics	Assembly	Recommend	ations	Why No Vector	orization?		
File: stencil	_v2.c:63	3 bench_stencil	_ver1\$omp	\$parallel_for	9 <b>62</b>				
Lin.			Source				Total Time	. %	Loop/Function Tir
61 ≞ whil	e(Elaps	edTime < TIME)	{						
62 #p	ragma o	mp parallel to	r						150452.000
62 1 40	$\Gamma (K \equiv I : K)$	<010+1:K++)3							159455.000
63 <sup>±</sup> fo	r(i=1)i	<pre><dim+1;i++){< pre=""></dim+1;i++){<></pre>							
63	r(j=1;j	<dim+1;j++){< td=""><td></td><td></td><td>Select</td><td>ed (Total Time):</td><td>0m</td><td>IS</td><td></td></dim+1;j++){<>			Select	ed (Total Time):	0m	IS	





•••				NoM	achine - NERSC			_
≝ ⊙ /global/cs File View He	elp	sd/tk	C	ORAM R	oofline*			
Welcome e0	× 00							
<b>b</b> Elapsed	time: 50	.40s 😽 🖸 Vect	orized 🗴 N	ot Vectorized 🍯 🛚	KL			OFF Sma
FILTER: All M	Iodules	✓ All Sources √	Loops An	d Functions 👻 All	Threads 👻			
Summary	🏷 Surv	ey & Roofline 📲	Refinemen	t Reports				
SC Perform	ance (G	FLOPS)	▶ Q ( <sup>®</sup> )	<ul> <li></li> <li><!--</th--><th>Use Single-Th</th><th>readed Roof</th><th>fs 🛛</th><th>Show Hierarchica</th></li></ul>	Use Single-Th	readed Roof	fs 🛛	Show Hierarchica
	L1 Band	width: 1.1e+4 GB/ width: 3542.01 GB	Vsec					DP Vector Add Pea Scalar Add Pea
1	L3 Bano	andwidth: 128.58	GB/sec?	-0	0			
0.66	DRAMU			8 1	[ <b>loop in bench_ste</b> Total Performance: Total L1 Arithmetic	ncil_ver2\$o 15.28 GFLO Intensity: 0.2	<b>mp\$para</b> PS 8 FLOP/I	allel_for@102 at ste Byte
s s	elf Elaps	0.05 ed Time: <b>0.008</b> s	Total Elap	sed Time: <b>9.964</b> s	Self Elapsed Time: Total Elapsed Time	0.008 s : 9.964 s		
Source Top	Down	Code Analytics	Assembly	Recommendation	s 🖣 Why No Vect	orization?		
File: stencil	_v2.c:1	03 bench_stenc	il_ver2\$om	p\$parallel_for@1	)2			
Lin.			Source			Total Time	. %	Loop/Function Tin
101 <sup>m</sup> while 102 #p	<pre>while(ElapsedTime &lt; TIME){     #pragma omp parallel for schedule(static,1)</pre>			16.002ms				
103 ⊞ fo	r(tile=	0;tile <jtiles*< td=""><td>Tiles; tile</td><td>e++){</td><td></td><td></td><td></td><td>159651.000</td></jtiles*<>	Tiles; tile	e++){				159651.000
104	int kLo	= 16*(tile/jT	iles);	Sel	ected (Total Time):	0m	S	
				501	leteu (Totur Time).			





	NoMachine - NERSC									
/global/cscratch1/sd/tk File View Help	DRAM Roofline*									
Welcome e000 X		•								
<ul> <li>Elapsed time: 50.40s</li> <li>FILTER: All Modules - A</li> <li>Summary Survey &amp;</li> </ul>	Vectorized Not Vectorized   MKL     All Sources   Loops And Functions   All Threads     Roofline     Refinement Reports	le° ۹ EL ADVISOR 2018								
은 Performance (GFLOP	PS) Reverse and the state of th	a 🔳								
1688.18	DP Vector FMA Peak: 168	3.18 GFLOPS								
→ L1 Bandwidth	DP Vector Add Peak: 42	2.07 GFL OPS								
1 3 Bandwidth	n: 1003.46 GB/sec Scalar Add Peak: 11	5.16 GFLOPS?								
	128.58 GB/sec									
0.66 0.05 Self Elapsed Time: 0.000 s Total Elapsed Time: 9.926 s 0.05 Self Elapsed Time: 9.926 s 0.05 Self Elapsed Time: 9.926 s 0.05 Self Elapsed Time: 9.926 s 0.000 s 0.0000 s 0.000 s 0.0000 s 0.000 s 0.000 s 0.0000 s 0.000 s 0										
Source Top Down Cod	le Analytics Assembly 🛛 Recommendations 📮 Why No Vectorization?									
File: stencil_v2.c:146 b	ench_stencil_ver3\$omp\$parallel_for@146									
Lin.	Source Total Time % Loop/Function Time	% Traits								
144 StartTime = omp	_get_wtime();									
145 <sup>—</sup> White (Etapsed in 146 <sup>—</sup> #pragma_omp_p	$me < IIME)$ { marallel for schedule(static.1) 43.998ms 159807.000ms	_								
147	<pre>.le<jtiles*ktiles;tile++){< pre=""></jtiles*ktiles;tile++){<></pre>	-								
	Selected (Total Time): 43.998ms									
	a Bamacs atk@cari05 hand0 224									
Advixe-gul	3 mareinaus-yuk@conos-bond0.224 metcon:									



	NoMachine - NERSC							
Sile View Help	DRAM Roofline*			©				
Welcome e000 x				-				
<ul> <li>Elapsed time: 50.40s</li> <li>FILTER: All Modules</li> <li>All Sources</li> <li>Summary</li> <li>Survey &amp; Roofline</li> </ul>	<ul> <li>ctorized O Not Vectorized O MKL</li> <li>✓ Loops And Functions ✓ All Threads ✓</li> <li>In Refinement Reports</li> </ul>		OFF Smart Mode <sup>®</sup> INTEL AD	visor 2018				
Performance (GFLOPS) 1688.18 L1 Bandwidth: 1.1e+4 G L2 Bandwidth: 3542.014	▶ Q (	eaded Roofs @	DP Vector FMA Peak: 1688.18 DP Vector Add Peak: 422.07 G	FLOPS				
L3 Bandwidth: 128.58 GB/sec?       Image: Scalar Add Peak: 115.16 GFLOPS         DRAM Bandwidth: 128.58 GB/sec?       Image: Scalar Add Peak: 115.16 GFLOPS         0.66       Image: Scalar Add Peak: 115.16 GFLOPS         Image: Scalar Add Peak: 115.16 GFLOPS       Image: Scalar Add Peak: 115.16 GFLOPS         Image: Scalar Add Peak: 115.16 GFLOPS       Image: Scalar Add Peak: 115.16 GFLOPS         Image: Image: Scalar Add Peak: 115.16 GFLOPS       Image: Scalar Add Peak: 115.16 GFLOPS         Image: I								
Source Top Down Code Analytics	Assembly Recommendations Why No Vector	orization?						
File: stencil_v2.c:193 bench_ster	ncil_ver4\$omp\$parallel_for@193							
Lin. 191 StartTime = omp_get_wtim 192 ⊞ while(ElapsedTime < TIME	Source e(); ){	Total Time %	Loop/Function Time %	Traits				
193 <sup>⊞</sup> #pragma omp parallel f	or schedule(static,1)	8.004ms	160161.000ms	1				
194 for(tile=0;tile <jtiles< th=""><th>*kTiles;tile++){ Selected (Total Time):</th><td>8.004ms</td><td></td><td></td></jtiles<>	*kTiles;tile++){ Selected (Total Time):	8.004ms						
🛃 🔚 🔛 Advixe-gui	₃ Semacs-gtk@cori05-bond0.224	Cori	:					



## Little's Law

### **Applied to Memory**

- Consider a CPU with 100GB/s of bandwidth and 100ns memory latency.
- Little's law states that we must **express 10KB of concurrency** (independent memory operations) to the memory subsystem to attain peak performance
- Solution #1: use multiple cores or threads to satisfy the requisite MLP.
- Solution #2: express the memory access pattern in a streaming fashion in order to engage the prefetchers.

#### **Applied to FPUs** consider a CPU with 2 FPU's each with a

- 4-cycle latency.
- Little's law states that we must express 8way ILP to fully utilize the machine.
- Solution #1: rely on OOO to find parallelism across loop iterations.
- Solution #2: unroll/jam the code to express 8 independent FP operations.
- Note, simply unrolling dependent operations (e.g. reduction) does not increase ILP. It simply amortizes loop overhead.

