Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm

Stephanie Labasan
Computer and Information Science
University of Oregon
23 November 2015

Overview

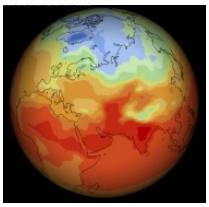
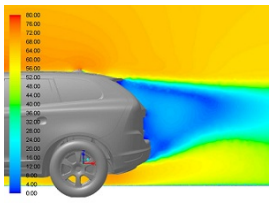
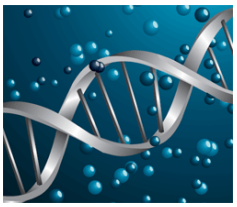
- **Motivation**: Power is becoming a leading design constraint in HPC
  - **Goal**: Save energy/power
- **Idea**: Reduce CPU clock frequency
  - Works because visualization is data-intensive
  - **Proposition for users**: Run **X%** slower, save **Y%** in energy/power
- **Study**: Consider a visualization algorithm (isosurfacing) and investigate which factors most affect X and Y

Outline

- Motivation & Background
- Strategy & Research Questions
- Experimental Overview
- Results
- Takeaways

High-Performance Computing

- Enabling technology for scientific discoveries
  - Weather prediction, design simulation, genomics research, medical imaging, and more!
- More cost-efficient to buy a supercomputer than building an actual experiment


Large Supercomputers

- Performance measured in FLOPs = floating point operations per second
  - 1 GigaFLOP = 1 billion FLOPs
  - 1 TeraFLOP = 1,000 GigaFLOPs
  - 1 PetaFLOP = 1 million GigaFLOPs → today
  - 1 ExaFLOP = 1 billion GigaFLOPs → expected 2020

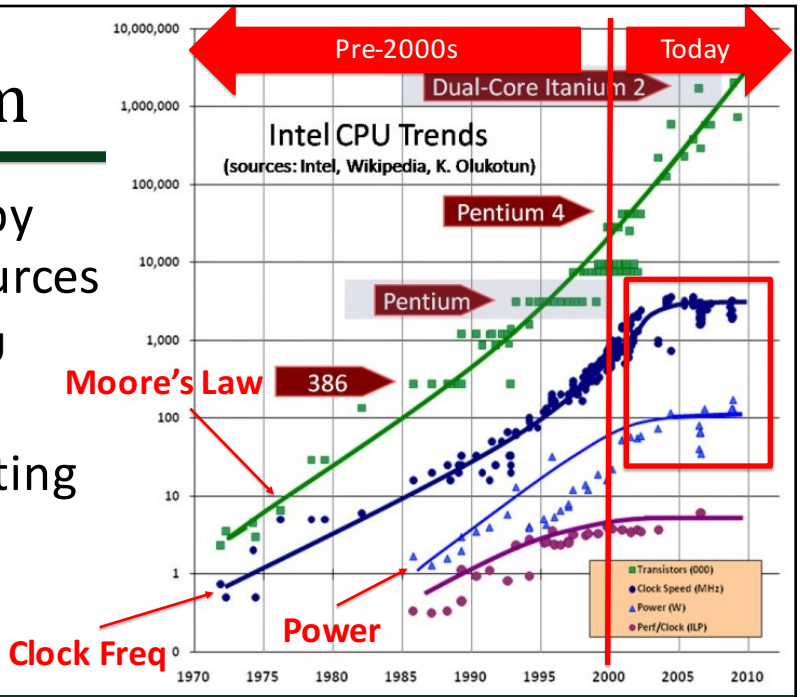
Power Costs

- 1 MW of power costs $1M per year
- Supercomputing centers are paying ~$10M
- As machines get larger, cost rises, unless we innovate power-efficient techniques
  - Techniques may come from HW, but SW as well

Rank	System	Cores	Rmax (PFLOPS)	Power (MW)
1	Tianhe-2, China	3.12M	33.9	17.8
2	Titan, ORNL	560K	17.6	8.2
3	Sequoia, LLNL	1.57M	17.2	7.9
4	K computer, Japan	705K	10.5	12.7

Power Problem

- No longer limited by available HW resources
  - Thermal heat ≈ CPU frequency
- System design shifting from hardware to software design



c/o http://www.extremetech.com/computing/

Energy vs. Power

Energy

- Units: Joules
- Total work done
- Monthly bill from electric company (e.g., $.11/kWh) – "power bill"

$$Energy = Power \cdot time$$

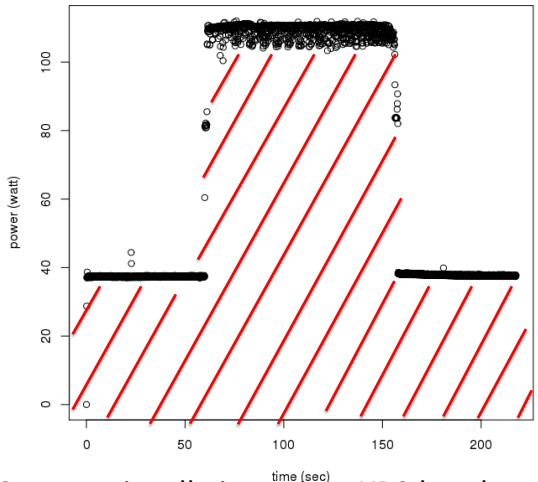
Power

- Units: Joules/s, Watts
- Rate of energy usage

$$Power = \frac{Energy}{time}$$

CPU Power Usage Varies Over Time

- Base power usage for idle state
- When more HW components are engaged (cache, vector units, etc.), power rises
- Energy usage would be the area under this power curve



Computationally-intensive HPC benchmark (Linpack) running on small cluster

Save Energy/Power?

Energy

- For a single application execution, minimize energy consumption ($$)
- Energy-to-solution
- Workstations, data centers

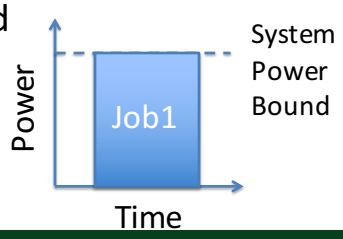
Power

- Reduce rate at which application consumes energy
- Save power != Save $$/energy
  - Run at 40W ↓ take 10s = 400J ↑
  - Run at 30W ↓ take 15s = 450J ↑
- Increase throughput (concurrent jobs)
- Supercomputers

Increasing Throughput

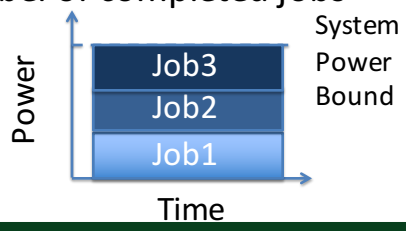
Scenario 1

- Launch a single job with maximum power allocation available
- No other concurrent jobs can be launched without exceeding the bound



Scenario 2

- Launch the same job at a lower power allocation, job runs longer
- Power available to launch other concurrent jobs → increase number of completed jobs



Department of Energy Perspectives on Exascale Challenges

Systems	2009	"~~2018~~" "2020"	Difference Today & ~~2018~~ 2020
System peak	2 Pflop/s	1 Eflop/s	O(1000)
Power	6 MW	~20 MW	~3
System memory	0.3 PB		
Node performance	125 GF		
Node memory BW	25 GB/s		
Node concurrency	12		
Total Node Interconnect BW	3.5 GB/s		
System size (nodes)	18,700		
MTTI	Days	O(1 day)	- O(10)

Systems	2009	"~~2018~~" "2020"
System peak	2 Pflop/s	1 Eflop/s
Power	6 MW	~20 MW

Until now, innovations in power savings have been hardware driven. However, to reach 1 Exaflop/20 MW, software improvements may be needed.

c/o P Beck

Power on Future HPC Systems

- Two scenarios:
  1. Machine's max power consumption <= 20 MW
     - i.e., even Linpack stays <= 20 MW
  2. Machine's max power consumption > 20 MW
     - Controls needed to enforce power usage at 20 MW cap
     - Power allocation would need to be coordinated by a central

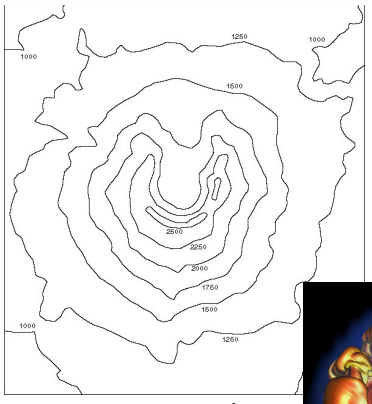
Our study is important for scenario (2), but also has cost saving implications for (1).

Last two years…

- Different visualization algorithms (volume rendering, particle advection, isosurfacing, …)
- Different implementations using various visualization software libraries, such as VTK, Dax, EAVL
- Building a "toolkit":
  - Power profiles → attribution
  - Tunable knobs to improve power efficiency

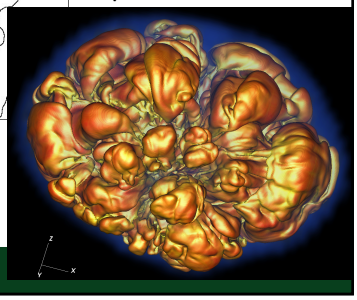
Isosurfacing

- Isolines: Represent where a field is constant
- Isosurface: 3D object, where every point is a constant value
- Method:
  - Iterate over all cells
  - Compare cell to isovalue (or range of isovalues)
    - Write valid cells into new data set
    - Throw out invalid cells



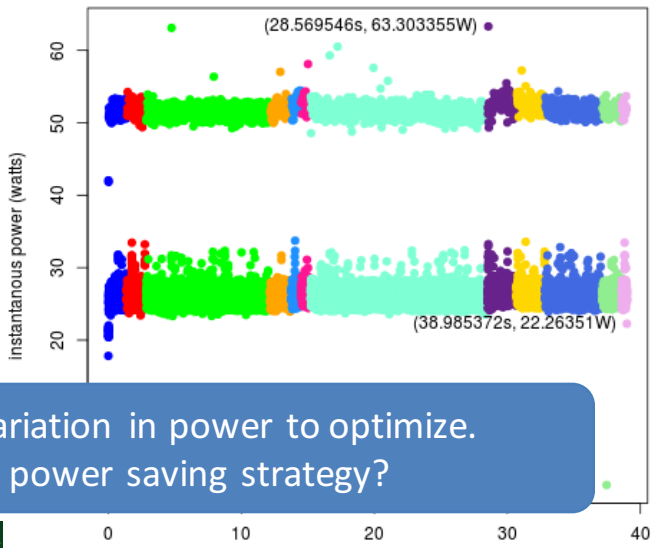
Contour Map: isolines

Temperature of Star Explosion: Isosurface



Isosurfacing Workload & Power

- Power profile primarily consistent with idle state (20W-30W)
- Power jumps to 50W presumably due to cache misses



Disappointing result: little variation in power to optimize. What about a different power saving strategy?

Outline

- Motivation & Background
- Strategy & Research Questions
- Experimental Overview
- Results
- Takeaways

Power Saving Technique: Reduce CPU Clock Frequency

- **Outcome**: Takes longer to run, but uses less power
- **Why**: Non-linear relationship between frequency and power → $P \propto f^3$
  - Reduced clock frequency results in less power consumption
  - but, subcomponents still consume power at the same rate

→ Will lead to power savings

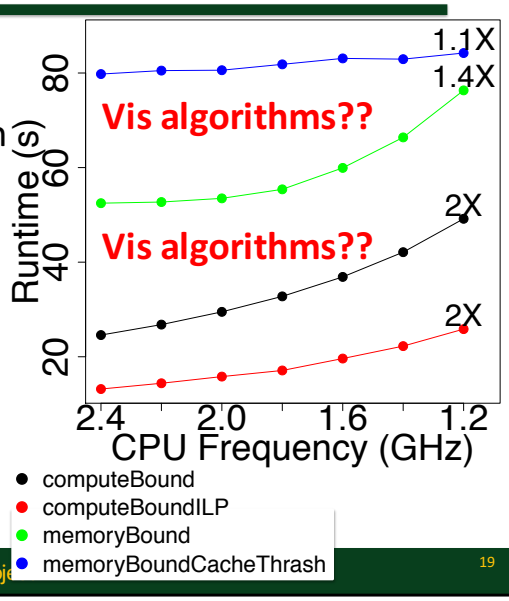
→ May lead to energy savings, may not

Data-Intensive Applications

- Architecture with separate clock frequencies for cache and compute unit (e.g., Intel Haswell)
  - Enables reduction of clock frequency without impacting cache
- Ideal outcome:
  - Reduce clock frequency
  - Achieve energy/power savings
  - Runtime does not change
- Some visualization algorithms are data-intensive

Benchmark Tests

- **computeBound**: Several multiplication ops on a single variable
- **computeBoundILP**: computeBound with instruction-level parallelism for pipelining
- **memoryBound**: Accesses an element in one array and writes it to another based on an index
- **memoryBoundCacheThrash**: memoryBound with randomized output locations, removing any benefit of locality



Research Questions

- **Data-intensity**: If you reduce the clock frequency by X, does the runtime increase by X? or less?
- **Energy savings**: If you are willing to run X% slower, you can save Y% in energy
- **Power savings**: If you are willing to run X% slower, you can save Z% in power

Outline

- Motivation & Background
- Strategy & Research Questions
- Experimental Overview
- Results
- Takeaways

Factors Studied

1. **Hardware architectures**
   - **2 options**

CPU1	CPU2
Workstation: Intel i7 4770K (Haswell)	Supercomputer: NERSC's Edison (Ivy Bridge)
4 HT cores @ 3.5 GHz	12 HT cores @ 2.4 GHz
32 GB memory @ 1600 MHz	64 GB memory @ 1866 MHz

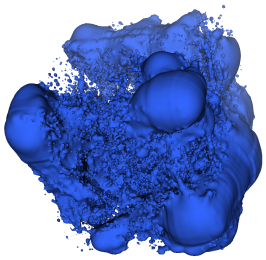
Factors Studied

1. Hardware architectures
2. **CPU clock frequency**
   - **varies**

CPU1	CPU2
Workstation: Intel i7 4770K (Haswell)	Supercomputer: NERSC's Edison (Ivy Bridge)
4 HT cores @ 3.5 GHz	12 HT cores @ 2.4 GHz
32 GB memory @ 1600 MHz	64 GB memory @ 1866 MHz
11 frequency steps (3.5GHz-1.6GHz, by=200MHz)	**7 frequency steps (2.4GHz-1.2GHz, by=200MHz)**
Linux cpufreq-utils tool	**ALPS aprun job launcher**

Factors Studied

1. Hardware architectures
2. CPU clock frequency
3. **Data set**
   - **8 options**



Enzo

Nek5000

Enzo	Nek5000
Cosmology simulation, structured, rectilinear grid to tetrahedrons	Thermal hydraulics simulation, unstructured, hexahedrons to tetrahedrons
Three data sets of varying sizes: Enzo-1M (1.13M), Enzo-10M (10.5M), Enzo-80M (83.9M)	Nek5000 (50M)
Same as above, but increase data-intensity: REnzo-1M, REnzo-10M, REnzo-80M	RNek5000
Isoval: 170	Isoval: 0.3

Factors Studied

1. Hardware architectures
2. CPU clock frequency
3. Data set
4. **Parallel programming model**
   - **2 options**
5. **Concurrency**
   - **4 options**

- **OpenMP**: Cores operate on a common data set, enabling opportunity for cache coordination
- **MPI**: Each core operates on a local copy of the data set, leading to uncoordinated cache accesses

Factors Studied

1. Hardware architectures
2. CPU clock frequency
3. Data set
4. Parallel programming model
5. Concurrency
6. **Isosurface implementation**
   - **2 options**

- **Baseline**: Our own implementation that only performs isosurfacing on tetrahedrons using linear interpolation.
- **General**: Implemented using VTK, specifically vtkContourFilter. Algorithm performs isosurfacing in a general way (many cell types, higher order elements, etc.).

Methodology

Factors Studied

1. Hardware architecture
2. CPU clock frequency
3. Data set
4. Parallel programming model
5. Concurrency
6. Isosurface implementation

- Study conducted in 6 phases
- Phase 1: Vary clock frequency on Baseline Implementation, 1 data set, OpenMP, max concurrency on workstation
- Each phase varies 1 factor to investigate effects

Performance Metrics

- Most collected with PAPI
  - Total instructions executed, total cycles, number of L3 cache misses
  - Derive instructions per cycle (IPC): metric of data-intensity (e.g., high IPC = compute-intensive, low IPC = data-intensive)
- Energy/power measurements
  - Total energy read from energy MSR
    - Derive average power consumed from total energy and program runtime
  - Cray power management infrastructure

Outline

- Motivation & Background
- Strategy & Research Questions
- Experimental Overview
- Results
- Takeaways

General Notation & Relationships

- Clock frequency:
  - F_D = nominal frequency
  - F_R = reduced frequency
- For a given application:
  - Runtime
    - T_D = time taken at freq F_D
    - T_R = time taken at freq F_R
  - Energy
    - E_D = energy used at freq F_D
    - E_R = energy used at freq F_R
  - Power

- $F_{rat} = F_D/F_R$
- $T_{rat} = T_R/T_D$
- $E_{rat} = E_D/E_R$
- $P_{rat} = P_D/P_R$

- If $T_{rat} < F_{rat}$, then data-intensive application (*e.g.,* does not slowdown proportional to clock frequency)

No need to be overwhelmed – we'll go over these in the next slides.

Phase 1 Results: Vary CPU Frequency

- Reducing clock frequency by 2.2X:
  - **Runtime**: increases by 1.9X
  - **Energy**: decreases by 1.4X
  - **Power**: decreases by 2.7X
- Implementation tends towards compute-intensive, but not as strongly as our compute-bound benchmark
- Favorable energy/power savings

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	1.29s	1X	74.3J	1X	57.4W	1X
3.3GHz	1.1X	1.32s	1X	69.4J	1.1X	52.6W	1.1X
3.1GHz	1.1X	1.38s	1.1X	66.7J	1.1X	48.2W	1.2X
2.9GHz	1.2X	1.42s	1.1X	63.4J	1.2X	44.8W	1.3X
2.7GHz	1.3X	1.50s	1.2X	61.5J	1.2X	40.9W	1.4X
2.5GHz	1.4X	1.62s	1.3X	60.9J	1.2X	37.5W	1.6X
2.3GHz	1.5X	1.78s	1.4X	53.7J	1.4X	30.1W	1.9X
2.1GHz	1.7X	1.93s	1.5X	53.8J	1.4X	27.9W	2.1X
2.0GHz	1.8X	1.95s	1.5X	52.1J	1.4X	26.8W	2.2X
1.8GHz	1.9X	2.13s	1.7X	51.1J	1.4X	24.1W	2.4X
1.6GHz	2.2X	2.40s	1.9X	51.4J	1.4X	21.4W	2.7X

Baseline Implementation, Enzo-10M, CPU1, OpenMP

Phase 2 Changes

- **Changes**: Execute our own isosurfacing algorithm on all 8 data sets
- **Goal**: Investigate how data-intensity and size affects our propositions

Phase 2 Results: Vary Data Set

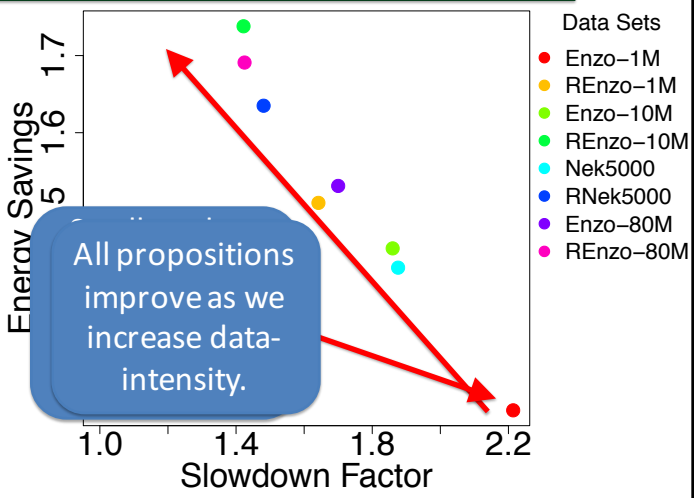
- Increase data-intensity of data set:
  - **Runtime**: increases by 1.4X (versus 1.9X in last slide)
    → More data-intensive than before
  - **Energy**: decreases by 1.7X (versus 1.4X in last slide)
  - **Power**: decreases by 2.5X (versus 2.7X in last slide) due to increased cache misses

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	2.95s	1X	142.7J	1X	48.4W	1X
3.3GHz	1.1X	3.05s	1X	134.8J	1.1X	44.2W	1.1X
3.1GHz	1.1X	3.01s	1X	124.3J	1.1X	41.3W	1.2X
2.9GHz	1.2X	3.33s	1.1X	122.3J	1.2X	36.8W	1.3X
2.7GHz	1.3X	3.23s	1.1X	109.3J	1.3X	33.8W	1.4X
2.5GHz	1.4X	3.22s	1.1X	99.6J	1.4X	30.9W	1.6X
2.3GHz	1.5X	3.48s	1.3X	93.4J	1.5X	26.8W	1.8X
2.1GHz	1.7X	3.49s	1.3X	88.0J	1.6X	25.2W	1.9X
2.0GHz	1.8X	3.79s	1.3X	88.3J	1.6X	23.3W	2.1X
1.8GHz	1.9X	3.79s	1.3X	82.2J	1.7X	21.7W	2.2X
1.6GHz	2.2X	4.19s	1.4X	82.1J	1.7X	19.6W	2.5X

Baseline Implementation, REnzo-10M, CPU1, OpenMP

Phase 2 Results: Vary Data Set

- Smallest data set suffers largest impact to runtime with minimal energy savings
- **Energy savings**: Improves for data sets with increased data-intensity
  - Cache misses differ across data sets



Baseline Implementation, CPU1, OpenMP

Phase 3 Changes

- **Changes**: Use MPI instead of OpenMP to implement our own isosurfacing algorithm and perform isosurfacing on all 8 data sets
- **Goal**: Investigate the effects of non-coordinated cache accesses on our algorithm

Parallel Model Differences

MPI

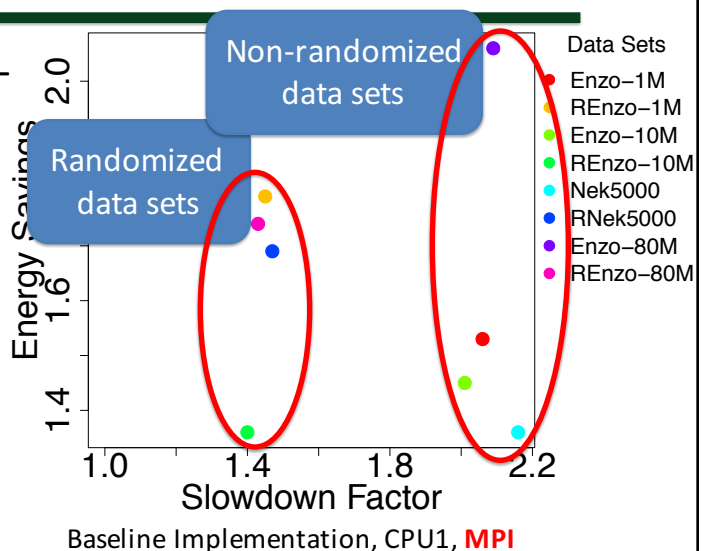
- Each rank loads in a local copy of the data set
- Each rank iterates over all cells in their local data

OpenMP

- Program loads data set
- Data set is split among threads, and each thread performs isosurface on a subset of cells
- Repeat, such that each thread iterates over all cells (same amount of work as MPI)

Phase 3 Results: Vary Programming Model

- Two clusters: Increase data-intensity of data set causes less impact to runtime
- **Energy savings**: Larger energy savings than OpenMP (1.7X vs. 2X)



Baseline Implementation, CPU1, **MPI**

Phase 3 Results: Vary Programming Model

- **Energy savings**: Better with MPI than with OpenMP, MPI uses more energy
  → MPI process is heavier weight than OpenMP thread
- **Power savings**: MPI uses more power than OpenMP, especially on randomized data set
  → MPI makes less efficient use of memory than OpenMP

Configuration	T	T_{rat}	E	E_{rat}	P	P_{rat}
MPI/Enzo-10M/3.5 GHz	1.08s	1X	74.5J	1X	69.2W	1X
MPI/Enzo-10M/1.6 GHz	2.08s	2X	51.8J	1.4X	24.9W	2.8X
MPI/REnzo-10M/3.5 GHz	3.46s	1X	179.5J	1X	51.9W	1X
MPI/REnzo-10M/1.6 GHz	4.62s	1.4X	95.5J	1.9X	20.7W	2.5X
OpenMP/Enzo-10M/3.5 GHz	1.29s	1X	74.3J	1X	57.4W	1X
OpenMP/Enzo-10M/1.6 GHz	2.40s	1.9X	51.4J	1.4X	21.4W	2.7X
OpenMP/REnzo-10M/3.5 GHz	2.95s	1X	142.7J	1X	48.4W	1X
OpenMP/REnzo-10M/1.6 GHz	4.19s	1.4X	82.1J	1.7X	19.6W	2.5X

38

Phase 4 Changes

- **Changes**: Increase the level of concurrency when executing the OpenMP and MPI implementations of our own isosurfacing algorithm
- **Goal**: Investigate the effects of increased memory requests between the two programming models

Phase 4 Results: Vary Concurrency

- **Power savings** @1 core: MPI and OpenMP save 2.5X
- **Power savings** @4 cores: MPI saves 2.8X and OpenMP saves 2.68X
- Increasing concurrency causes more memory requests, leading to memory BW saturation (less instructions executed) and increased L3 misses

Configuration	Time	Energy	Power	IPC	L3 Miss Rate
MPI/1/3.5 GHz	0.90s	24.4J	26.9W	2.37	5652
MPI/1/1.6 GHz	2.0s	21.1J	10.8W	2.41	3788
OpenMP/1/3.5 GHz	0.83s	19.9J	23.9W	2.06	1697
OpenMP/1/1.6 GHz	1.74s	16.7J	9.6W	2.11	931
MPI/4/3.5 GHz	0.96s	69.5J	72.2W	2.07	10476
MPI/4/1.6 GHz	2.02s	51.2J	25.3W	2.37	3456
OpenMP/4/3.5 GHz	1.29s	74.3J	57.4W	1.51	3351
OpenMP/4/1.6 GHz	2.40s	51.1J	21.4W	1.89	1027

BaselineImplementation, Enzo-10M, CPU1, MPI/OpenMP

Phase 5 Changes

- **Changes**: Implement the isosurfacing algorithm using VTK's contour filter
- **Goal**: Investigate the effects of using a different implementation, specifically an implementation using general-purpose visualization library that is designed for diverse usage (e.g., cell types, higher order elements, etc.)

Algorithm Implementation Differences

Baseline Implementation

- Pre-compute correct answers for all cases and store in a lookup table
- For each cell in data set, identify which case it is in
- Use lookup table to generate isosurface

General Implementation – VTK

- Visualization Toolkit
- Open source visualization software toolkit
- Enables computer graphics, visualization, and image processing

```
vtkContourFilter *cf;
cf->SetNumberOfContours(1);
cf->SetValue(170,isoval);
cf->Update();
```


Phase 5 Results: Vary Algorithm Implementation

- **Compute-intensive**: F_{rat} and T_{rat} are highly correlated
- Number of instructions issued differs between implementations (102 billion vs. 7 billion)
- Algorithm shifts from data- to compute-intensive and propositions become less favorable

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	16.06s	1X	1056J	1X	65.8W	1X
3.3GHz	1.1X	16.57s	1X	992J	1.1X	59.9W	1.1X
3.1GHz	1.1X	17.64s	1.1X	950J	1.1X	53.9W	1.2X
2.9GHz	1.2X	19.00s	1.3X	928J	1.1X	48.8W	1.3X
2.7GHz	1.3X	20.85s	1.3X	914J	1.2X	43.9W	1.5X
2.5GHz	1.4X	21.82s	1.4X	876J	1.2X	40.1W	1.6X
2.3GHz	1.5X	24.01s	1.4X	784J	1.3X	32.7W	2X
2.1GHz	1.7X	26.09s	1.7X	763J	1.4X	29.3W	2.2X
2.0GHz	1.8X	27.43s	1.7X	768J	1.4X	28.0W	2.4X
1.8GHz	1.9X	30.67s	1.9X	764J	1.4X	24.9W	2.6X
1.6GHz	2.2X	34.17s	2.1X	756J	1.4X	22.1W	3X

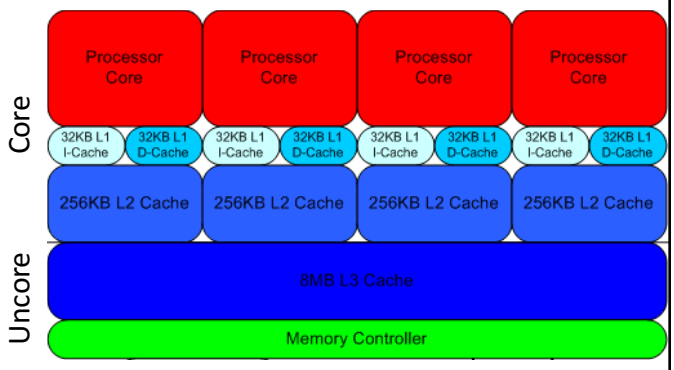
General Implementation, Enzo-10M, CPU1, OpenMP

Phase 6 Changes

- **Changes**: Execute our own implementation with MPI on a different processor architecture
  - Shared clock frequency between cache and compute unit
  - In previous phases, clock frequency was decoupled
- **Goal**: Investigate the effects of alternate hardware architectures

Decoupled Clock Frequency

- **Core**: Components of CPU involved in executing instructions
- **Uncore**: Components closely connected to the core to reduce access latency
- Intel architectures & clock frequency
  - Ivy Bridge: shared
  - Haswell: separate



Phase 6 Results: Vary Hardware Architecture

- Single clock frequency affects compute units AND shared L3 cache rates on Intel Ivy Bridge
- Propositions now unfavorable

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
2.4	1X	2.265	1X	549	1X	242.4	1X
2.2	1.1X	2.479	1.1X	558	1X	225	1.1X
2.0	1.2X	2.695	1.2X	571	1X	211.9	1.1X
1.8	1.3X	3.024	1.3X	573	1X	189.5	1.3X
1.6	1.5X	3.385	1.5X	631	0.9X	186.4	1.3X
1.4	1.7X	3.836	1.7X	668	0.8X	174.1	1.4X
1.2	2X	4.466	2X	697	0.8X	156	1.6X

Baseline Implementation, Enzo-10M, CPU2, MPI

Outline

- Motivation & Background
- Strategy & Research Questions
- Experimental Overview
- Results
- Takeaways

Study Takeaways

- Isosurfacing is sufficiently data-intensive to gain energy/power savings by reducing the clock frequency
- Tradeoffs between runtime and energy/power become more favorable when:
  - Increasing data set complexity (e.g., size, irregular access patterns)
  - Heavily stressing memory subsystem (e.g., concurrency)
- General-purpose visualization software shifts instruction counts, resulting in less favorable tradeoffs between runtime and energy/power savings

Future Work

- Applying factors to more visualization algorithms
  - i.e., particle advection, volume rendering
- Investigate impacts of different architectures
  - i.e., many-core, GPUs

Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm

Stephanie Labasan
slabasan@cs.uoregon.edu
http://cdux.cs.uoregon.edu

Thanks!

Phase 4 Results: Vary Concurrency*

- **Power savings** @1 core: MPI and OpenMP save 2.5X
- **Power savings** @4 cores: MPI saves 2.8X and OpenMP saves 2.68X
- Increasing concurrency causes more memory requests, leading to memory BW saturation (less instructions executed)*

*performance metrics in paper

Configuration	Time	Energy	Power
MPI/1/3.5 GHz	0.90s	24.4J	26.9W
MPI/1/1.6 GHz	2.0s	21.1J	10.8W
OpenMP/1/3.5 GHz	0.83s	19.9J	23.9W
OpenMP/1/1.6 GHz	1.74s	16.7J	9.6W
MPI/4/3.5 GHz	0.96s	69.5J	72.2W
MPI/4/1.6 GHz	2.02s	51.2J	25.3W
OpenMP/4/3.5 GHz	1.29s	74.3J	57.4W
OpenMP/4/1.6 GHz	2.40s	51.1J	21.4W

Baseline Implementation, Enzo-10M, CPU1, OpenMP/MPI

Power Problem*

- No longer limited by available HW resources
  - Thermal heat ≈ CPU frequency
- System design shifting from hardware to software design