Dagstuhl Perspectives Workshop on
Co-Design of Systems and Applications for Exascale
May 21-25, 2012

INSTITUTE OF COMPUTING TECHNOLOGY

Relating Exascale to Parallelism, Power and Energy

Zhiwei Xu
Institute of Computing Technology (ICT)
Chinese Academy of Sciences (CAS)
www.ict.ac.cn
zxu@ict.ac.cn

Supported in part by the CAS Ternary Computing project, the China Cloud Computing project, and the China Basic Research (973) program

A Co-Design Challenge: Design Space Explosion

- Methods widely used: benchmarking, meticulous modeling (and simulation)
  - Vladimir Getov, Adolfy Hoisie, Harvey J. Wasserman: Codesign for Systems and Applications: Charting the Path to Exascale Computing. *IEEE Computer* Nov. 2011
- Difficult and time consuming to explore a huge design space (a μP example)
  - 26 benchmarks and cycle-by-cycle simulation; each simulation needs one week
  - **Performance can improve up to 13 times**
  - The microprocessor has 10 adjustable parameters, leading to **70 million options**

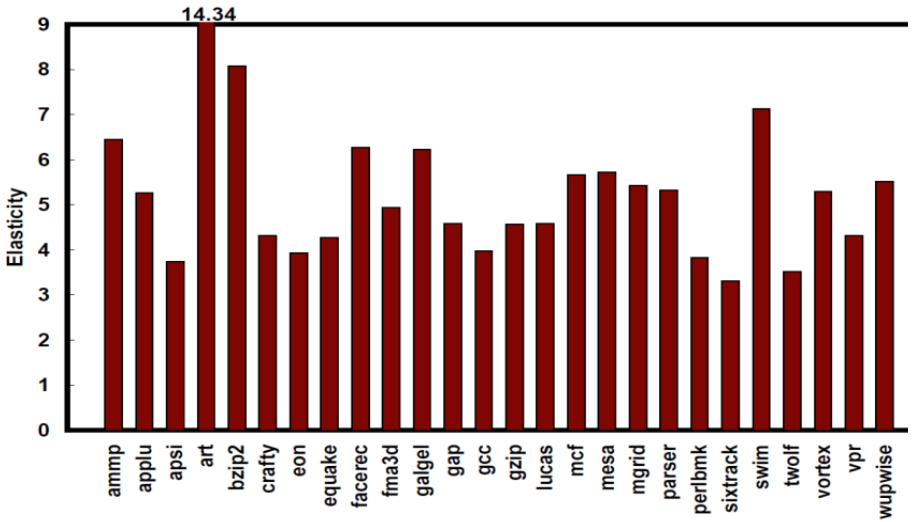


Table 2: Investigated microprocessor design space.

Abbr.	Parameter	Value
WIDTH	Fetch/Issue/Commit Width	2,4,6,8
FUNIT	FPALU/FPMULT Units	2,4,6,8
IUNIT	IALU/IMULT Units	2,4,6,8
L1IC	L1-ICache	8,16,32,64,128,256KB
L1DC	L1-DCache	8,16,32,64,128,256KB
L2UC	L2-UCache	256,512,1024,2048,4096KB
ROB	ROB size	16-256 with a step of 16
LSQ	LSQ size	8-128 with a step of 8
GSHARE	GShare size	1,2,4,8,16,32K
BTB	BTB size	512,1024,2048,4096
Total	**10 parameters**	**70,778,880 Options**

Q. Guo, T. Chen, Y. Chen, Z. Zhou, W Hu, Z. Xu, Effective and Efficient Microprocessor Design Space Exploration Using Unlabeled Design Configurations, *IJCAI* 2011
An expanded version accepted to appear in *ACM Transactions on Intelligent Systems and Technology*

A Complementary Approach

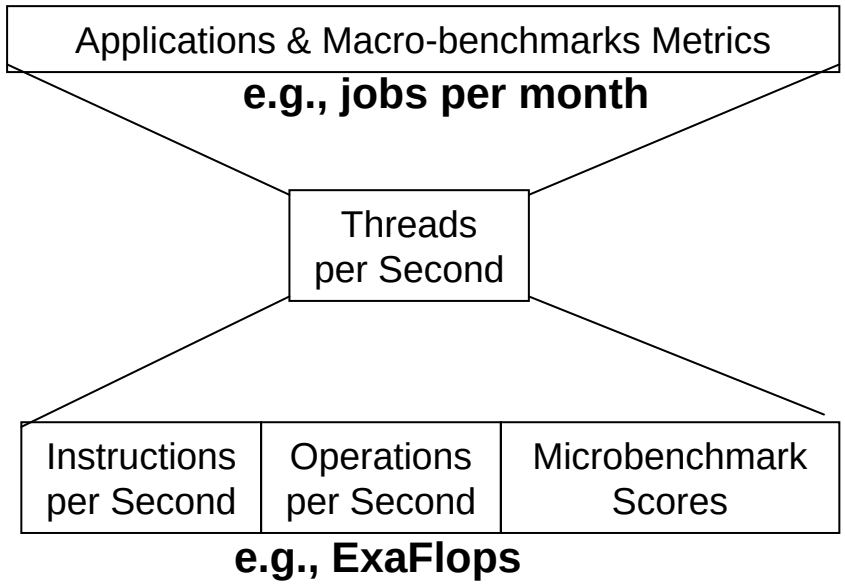
- Learn from Amdahl's Law, Little's Law, the CPI formula, and other scaling "laws"
  - All are simple: captured sophisticated common sense
- Hennessey and Patterson's CPI formula has helped the design of processor micro-architecture

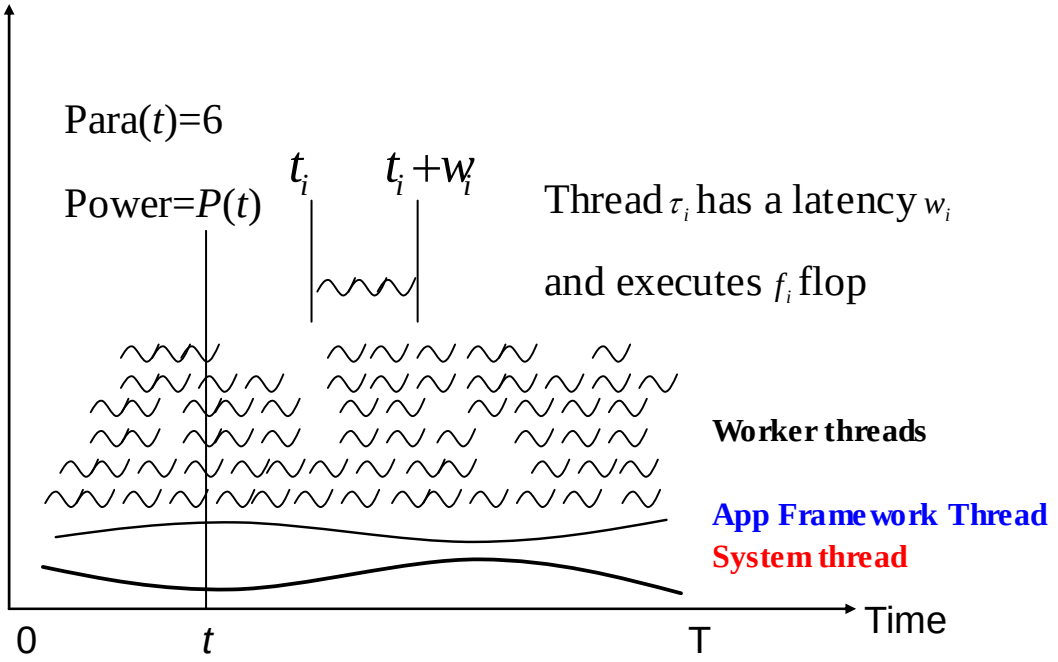
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

  - Example: Helps articulate micro-architecture classes
    - CPI > 1 → CISC architecture
    - CPI = 1 → RISC architecture
    - CPI < 1 → Superscalar architecture
- Not connected to power and energy

Borrow from Little's Law and the Internet Hourglass

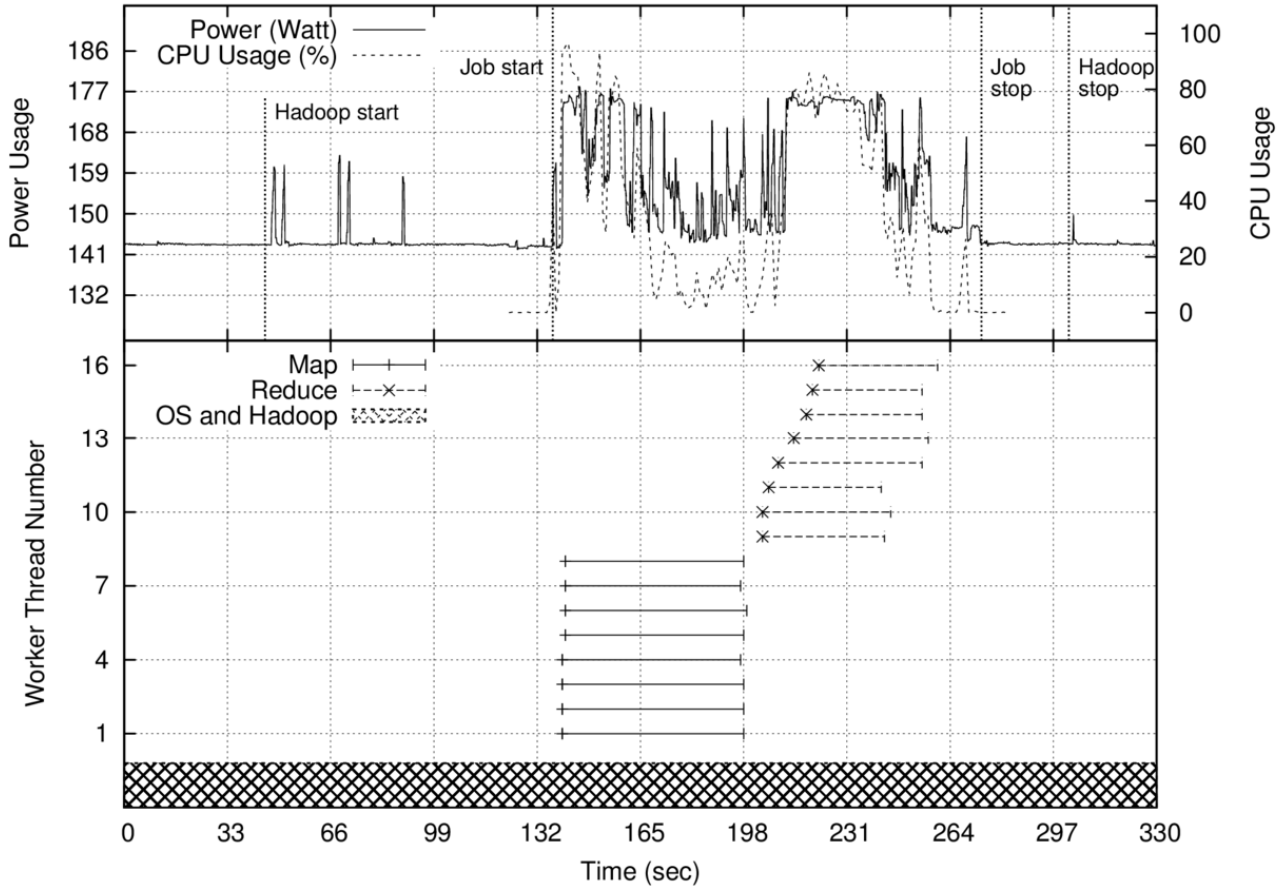
- Focus on "threads per second" as a proxy of the performance goals
  - Subject to latency, power, energy constraints
  - A thread is **a schedulable sequence of instruction executions with its own program counter**
    - POSIX thread, HW thread, Java thread, CUDA thread of GPU, Hadoop task, etc.
    - "Threads per second" serves as the neck of the performance metrics hourglass
- Need benchmarks to characterize app workloads and flop/thread
  - ExaFlops → a billion active threads (Thomas Sterling)





An Experiment with Data Computing

- Sorting 1 GB data with Hadoop on a 4-core computer
- 16 worker threads, 136 system processes and 2 Hadoop processes
- *N*=154, but only 16 tasks do the real app work
- 143/179=80% power “wasted”, even when utilization approaches 0%



Assumptions and Observations

- Assume N threads $\{\tau 1,\ldots,\tau N\}$ are executed in a computer system in time period [0, T], where
  - power and energy are additive; inactive threads consume no power
- Definitions of some average quantities
  - **Throughput** λ: threads per second, averaged over [0, T]
  - **Parallelism** ***L***: number of active threads, averaged over [0, T]
  - **Latency** ***W***: latency of a thread, averaged over $\{\tau 1,\ldots,\tau N\}$
  - **Power** ***P***: Watts consumed by the system, averaged over [0, T]
  - **Energy** ***E***: Joules consumed by a thread, averaged over $\{\tau 1,\ldots,\tau N\}$
- Observations
  - Little's Law: $\lambda = L / W$
  - New observations
    - $\lambda = P / E$
    - $\lambda = L \times (E/W) \times (1/E)$
    - **Throughput = Parallelism × Watts per thread × Threads per Joule**

Connecting to ExaFlops

- Definitions
  - **Work *F***: flops per thread, averaged over {τ1,…,τ*N*}

$$F = \sum_{i=1}^{N} f_i / N$$

  - **Speed S**:

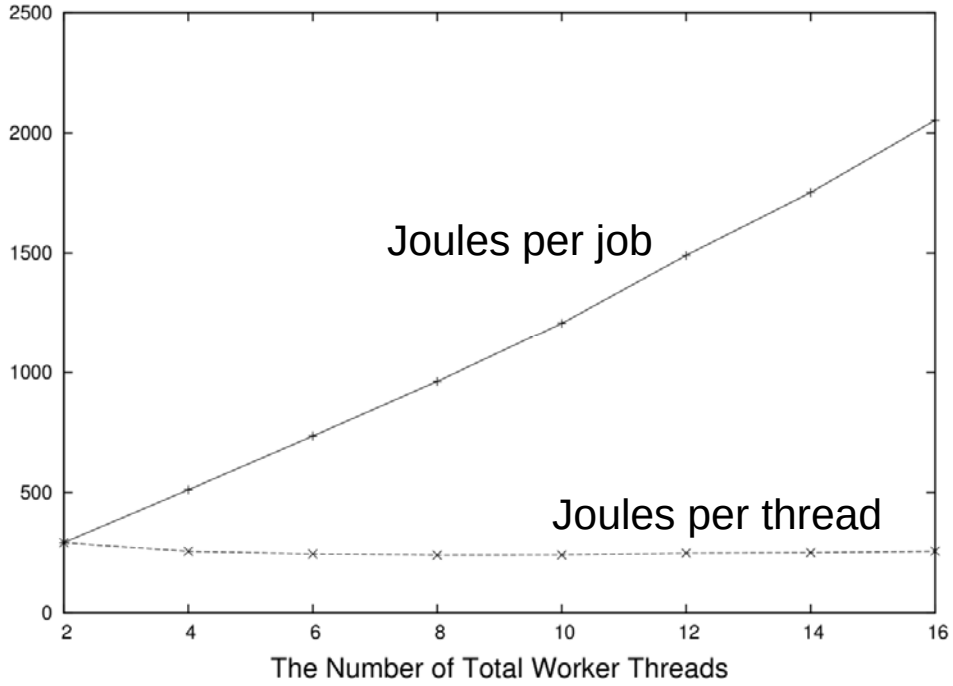
$$S = \sum_{i=1}^{N} f_i / T$$

- Observation
  - ***S* = *F* × λ = *L* × *F* × (*E*/*W*) × (1/*E*)**
  - **Speed = Parallelism × Work × Watts per thread × Threads per Joule**
  - **Eflops ?=? billion threads × billion flops per thread × ?? × ??**

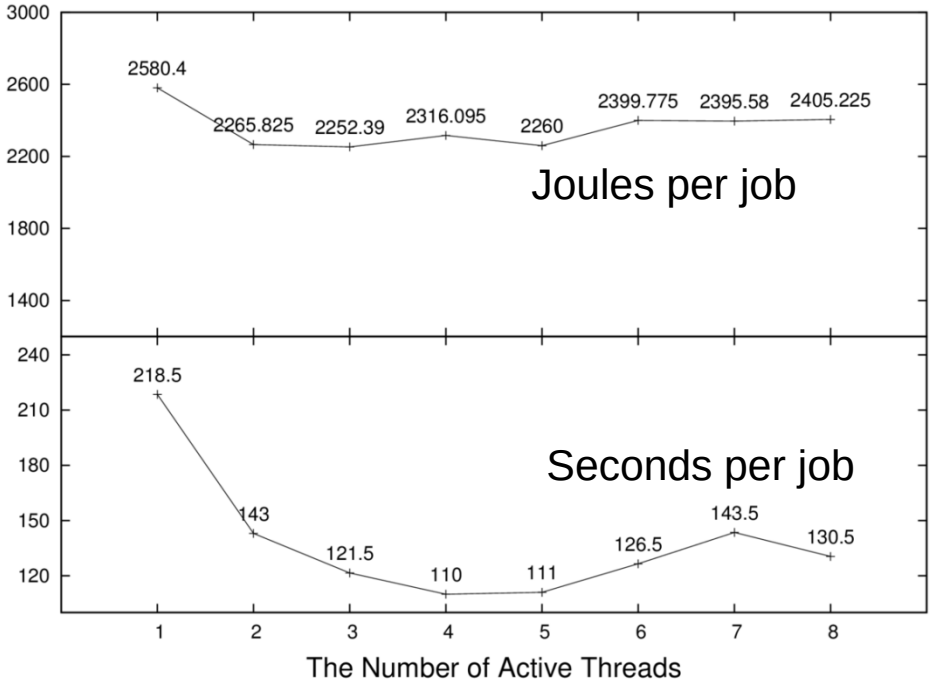
Summary

- Need simple rules of thumb
  - That are close to reality
    - Are power (and energy) additive?
- Billion-thread computer with elastic processors

Scaled Workloads (128 MB to 1 GB)



Fixed Workloads (1 GB)



zxu@ict.ac.cn

A Billion Thread Parallelism by 2020 for High-End Datacenter Computers

- $\lambda = L \times (E/W) \times (1/E)$
  Throughput = Volume × Watts per thread × Threads per Joule
- For power and energy efficient architecture design
  - Maximize *L* with good enough *W* for user experience
  - Architecture design aims to increasing *L* and 1/*E*, while technology advances controlling *E*/*W*
- How big "peak *L*" was/is/will be
  - 2000: kilo threads
  - 2010: million threads
  - 2020: billion threads
- Need 100-1000x improvement when power < 20 MW

Attributes of a DCC	2010	2020
Daily PV (billion)	4-7	20-100
Active threads per PV	1000	10,000
Peak-to-average ratio	2-10	2-15
Peak volume	millions	~1 billion

Connecting to ExaFlops

- Definitions
  - **Throughput** $\lambda = N / T$: threads per second, averaged over [0, *T*]
  - **Parallelism *L***: number of active threads, averaged over [0, *T*]
  - **Latency *W***: latency of a thread, averaged over {τ1,…,τ*N*}
  - **Energy *E***: Joules consumed by a thread, averaged over {τ1,…,τ*N*}
  - **Work *F***: flops per thread, averaged over {τ1,…,τ*N*}
  - **Speed S**: $$F = \sum_{i=1}^{N} f_i / N$$
- Observation $$S = \sum_{i=1}^{N} f_i / T$$

$$S = \sum_{i=1}^{N} f_i / T = (\sum_{i=1}^{N} f_i / N) \times (N / T) = F \times \lambda$$

  - ***S* = *F* × λ**
  - ***S* = *L* × *F* × (*E*/*W*) × (1/*E*)**