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)
- 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.

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDTH</td>
<td>Fetch/Issue/Commit Width</td>
<td>2,4,6,8</td>
</tr>
<tr>
<td>FUNIT</td>
<td>FPALU/FPMULT Units</td>
<td>2,4,6,8</td>
</tr>
<tr>
<td>IUNIT</td>
<td>IALU/IMULT Units</td>
<td>2,4,6,8</td>
</tr>
<tr>
<td>L1HC</td>
<td>L1-I-Cache</td>
<td>8,16,32,64,128,256KB</td>
</tr>
<tr>
<td>L1DC</td>
<td>L1-D-Cache</td>
<td>8,16,32,64,128,256KB</td>
</tr>
<tr>
<td>L2UC</td>
<td>L2-U-Cache</td>
<td>256,512,1024,2048,4096KB</td>
</tr>
<tr>
<td>ROB</td>
<td>ROB size</td>
<td>16-256 with a step of 16</td>
</tr>
<tr>
<td>LSQ</td>
<td>LSQ size</td>
<td>8-128 with a step of 8</td>
</tr>
<tr>
<td>GSHARE</td>
<td>GShare size</td>
<td>1,2,4,8,16,32K</td>
</tr>
<tr>
<td>BTB</td>
<td>BTB size</td>
<td>512,1024,2048,4096</td>
</tr>
<tr>
<td>Total</td>
<td>10 parameters</td>
<td>70,778,880 Options</td>
</tr>
</tbody>
</table>

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

• Hennessy 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 $\lambda$: 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 $\times$ Watts per thread $\times$ Threads per Joule
Connecting to ExaFlops

• Definitions
  – **Work** \( F \): flops per thread, averaged over \( \{\tau_1, \ldots, \tau_N\} \)
    \[
    F = \frac{1}{N} \sum_{i=1}^{N} f_i
    \]
  – **Speed** \( S \):
    \[
    S = \frac{1}{T} \sum_{i=1}^{N} f_i
    \]

• Observation
  – \( S = F \times \lambda = L \times F \times \left(\frac{E}{W}\right) \times \left(\frac{1}{E}\right) \)
    – Speed = Parallelism \( \times \) Work \( \times \) Watts per thread \( \times \) Threads per Joule
    – Eflops ?=? billion threads \( \times \) billion flops per thread \( \times \) ?? \( \times \) ??
Summary

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

![Graphs showing Joules per job and Joules per thread for Scaled Workloads and Fixed Workloads.](image)
谢谢！
Thank you!

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 $\times$ Watts per thread $\times$ 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

<table>
<thead>
<tr>
<th>Attributes of a DCC</th>
<th>2010</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daily PV (billion)</td>
<td>4-7</td>
<td>20-100</td>
</tr>
<tr>
<td>Active threads per PV</td>
<td>1000</td>
<td>10,000</td>
</tr>
<tr>
<td>Peak-to-average ratio</td>
<td>2-10</td>
<td>2-15</td>
</tr>
<tr>
<td>Peak volume</td>
<td>millions</td>
<td>~1 billion</td>
</tr>
</tbody>
</table>
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 $\{\tau_1, \ldots, \tau_N\}$
  – Energy $E$: Joules consumed by a thread, averaged over $\{\tau_1, \ldots, \tau_N\}$
  – Work $F$: flops per thread, averaged over $\{\tau_1, \ldots, \tau_N\}$

– Speed $S$:

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

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

• Observation

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

– $S = F \times \lambda$
– $S = L \times F \times (E/W) \times (1/E)$