Chapter 3 – Instruction-Level Parallelism and its Exploitation (Part 5)

- ILP vs. Parallel Computers
- Dynamic Scheduling (Section 3.4, 3.5)
- Dynamic Branch Prediction (Section 3.3, 3.9, and Appendix C)
- Hardware Speculation and Precise Interrupts (Section 3.6)
- Multiple Issue (Section 3.7)
- Static Techniques (Section 3.2, Appendix H)
- Limits and Benefits of ILP (Older editions and Section 3.12)
- Multithreading (Section 3.11)
- Putting it Together (Mini-projects)

Limits of ILP

- How much can ILP buy us?
- Limits studies make optimistic assumptions to find the limit for ILP
  - But may miss impact of compiler, future advances
- A highly optimistic study [Wall'93]
  - Infinite number of physical registers (no register WAW, WAR)
  - Infinite number of in-flight instructions
  - Perfect branch prediction
  - Perfect memory address alias analysis
  - Single cycle FU
  - Single cycle memory (perfect caches)

Limits of ILP (contd.)

(This and next four figures are from an **old** edition of the book)

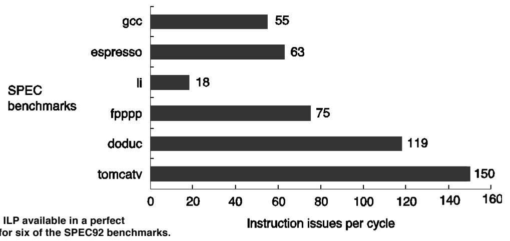
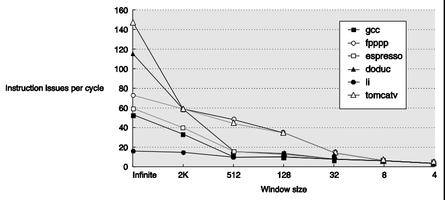


Figure 3.35 ILP available in a perfect processor for six of the SPEC92 benchmarks. The first three programs are integer programs, and the last three are floating-point programs. The floating-point programs are loop-intensive and have large amounts of loop-level parallelism.

Limits of ILP – Impact of Optimistic Assumptions

- Limiting Instruction window size
  - Finding dependences among n instr requires n^2 comparisons
  - 2000 instructions implies 4 million comparisons!
  - Following use 2K window and 64 issue limit

Figure 3.36 The effects of reducing the size of the window. The window is the group of instructions from which an instruction an execute. The start of the window is the earliest uncompleted instruction (remember that instructions complete in one cycle), and the last instruction in the window is determined by the window size. The instructions in the window are obtained by perfectly predicting branches and selecting instructions until the window is full.



Limits of ILP – Impact of Optimistic Assumptions

Realistic branch prediction

- No charge for mispredictions
- Following use tournament predictor

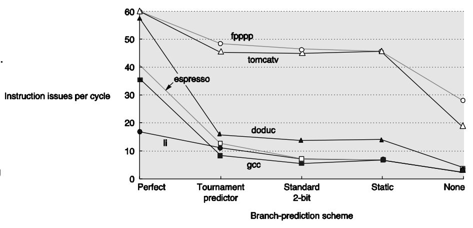


Figure 3.38 The effect of branch-prediction schemes. This graph shows the impact of going from a perfect model of branch prediction (all branches predicted correctly arbitrarily far ahead), to various dynamic predictors (selective and 2-bit), to compile time, profile-based prediction, and finally to using no predictor. The predictors are described precisely in the text.

Limits of ILP – Impact of Optimistic Assumptions

Finite registers

- Following uses 256 int and 256 fp for renaming

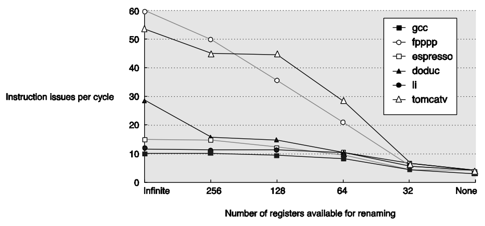


Figure 3.41 The effect of finite numbers of registers available for renaming. Both the number of FP registers and the number of GP registers are increased by the number shown on the x-axis. The effect is most dramatic on the FP programs, although having only 32 extra GP and 32 extra FP registers has significant impact on all the programs. As stated earlier, we assume a window size of 2K entries and a maximum issue width of 64 instructions. "None" implies no extra registers available.

Limits of ILP – Impact of Optimistic Assumptions

Imperfect memory alias analysis

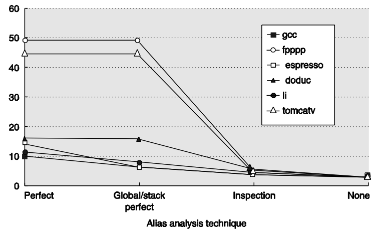


Figure 3.43 The effect of various alias analysis techniques on the amount of ILP. Anything less than perfect analysis has a dramatic impact on the amount of parallelism found in the integer programs, and global/stack analysis is perfect (and unrealizable) for the FORTRAN programs. As we said earlier, we assume a maximum issue width of 64 instructions and a window of 2K instructions.

But Limits Studies may be Pessimistic!

For most optimistic study

- WAR and WAW hazards through memory
- Unnecessary dependences (e.g., loop iteration count)
- Overcoming data flow limit – value prediction

For more realistic studies

- Address value prediction and speculation
- Speculating on multiple paths

Real Systems: Two Out-of-order i7 Processors

Resource	i7 920 (Nehalem)	i7 6700 (Skylake)
Micro-op queue (per thread)	28	64
Reservation stations	36	97
Integer registers	NA	180
FP registers	NA	168
Outstanding load buffer	48	72
Outstanding store buffer	32	56
Reorder buffer	128	256

Figure 3.39 The buffers and queues in the first generation i7 and the latest generation i7. Nehalem used a reservation station plus reorder buffer organization. In later microarchitectures, the reservation stations serve as scheduling resources, and register renaming is used rather than the reorder buffer; the reorder buffer in the Skylake microarchitecture serves only to buffer control information. The choices of the size of various buffers and renaming registers, while appearing sometimes arbitrary, are likely based on extensive simulation.

9

Two Out-of-order i7 Processors

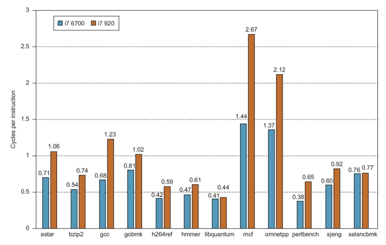


Figure 3.40 The CPI for the SPECCPUint2006 benchmarks on the i7 6700 and the i7 920. The data in this section were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University.

10

Out-of-order i7 vs. In-order Atom

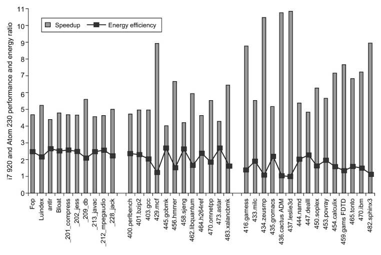


Figure 3.43 The relative performance and energy efficiency for a set of single-threaded benchmarks shows the i7 920 is 4 to over 10 times faster than the Atom 230 but that it is about 2 times *less* power-efficient on average! Performance is shown in the columns as i7 relative to Atom, which is execution time (i7)/execution time (Atom). Energy is shown with the line as Energy (Atom)/Energy (i7). The i7 never beats the Atom in energy efficiency, although it is essentially as good on four benchmarks, three of which are floating point. The data shown here were collected by Esmaeilzadeh et al. (2011). The SPEC benchmarks were compiled with optimization using the standard Intel compiler, while the Java benchmarks use the Sun (Oracle) Hotspot Java VM. Only one core is active on the i7, and the rest are in deep power saving mode. Turbo Boost is used on the i7, which increases its performance advantage but slightly decreases its relative energy efficiency.

11

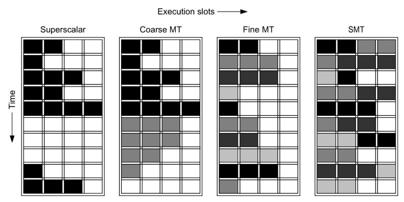
Multithreading: Instruction + Thread Level Parallelism

Often superscalar instruction slots are wasted

Why not use them for other threads?

- **Multithreading**
  - Coarse-grained
  - Fine-grained
  - Simultaneous multithreading (SMT) or hyperthreading
- (Vs. multiprocessing)

Multithreading: Instruction + Thread Level Parallelism



Vs. Multiprocessing?

Figure 3.31 How four different approaches use the functional unit execution slots of a superscalar processor. The horizontal dimension represents the instruction execution capability in each clock cycle. The vertical dimension represents a sequence of clock cycles. An empty (white) box indicates that the corresponding execution slot is unused in that clock cycle. The shades of gray and black correspond to four different threads in the multithreading processors. Black is also used to indicate the occupied issue slots in the case of the superscalar without multithreading support. The Sun T1 and T2 (aka Niagara) processors are fine-grained, multithreaded processors, while the Intel Core i7 and IBM Power7 processors use SMT. The T2 has 8 threads, the Power7 has 4, and the Intel i7 has 2. In all existing SMTs, instructions issue from only one thread at a time. The difference in SMT is that the subsequent decision to execute an instruction is decoupled and could execute the operations coming from several different instructions in the same clock cycle.

SMT Speedup & Energy Efficiency: 1 vs. 4 threads

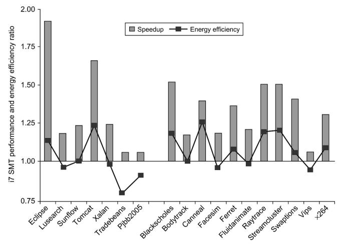


Figure 3.33 The speedup from using multithreading on one core on an i7 processor averages 1.28 for the Java benchmarks and 1.31 for the PARSEC benchmarks (using an unweighted harmonic mean, which implies a workload where the total time spent executing each benchmark in the single-threaded base set was the same). The energy efficiency averages 0.99 and 1.07, respectively (using the harmonic mean). Recall that anything above 1.0 for energy efficiency indicates that the feature reduces execution time by more than it increases average power. Two of the Java benchmarks experience little speedup and have significant negative energy efficiency because of this issue. Turbo Boost is off in all cases. These data were collected and analyzed by Esmaeilzadeh et al. (2011) using the Oracle (Sun) HotSpot build 16.3-b01 Java 1.6.0 Virtual Machine and the gcc v4.4.1 native compiler.