Parallel Computing

Institut für Informatik der Ludwig-Maximilians-Universität München
Vorlesung und Übung im Wintersemester 2017/18
Prof. Dr. D. Kranzlmüller, Dr. K. Fürlinger

Work sheet 2

Due Thursday, 09. November 2017, 12:00 pm on Uniworx.

Please submit your answers **in a single PDF document** if not stated otherwise.

---

1. **Pipelining I**
   Assuming we designed a MIPS processor where the pipeline stages have the following durations:

   IF = 20ns
   ID = 10ns
   EX = 20ns
   MEM = 35ns
   WB = 10ns

   - What is the processor's maximum clock frequency?
   - How could we achieve a performance gain in the next processor generation?
   - A competitor developed a processor with 50 pipeline stages and advertises it to yield a 10x performance increase compared to our 5-stage MIPS processor.
     How can this statement be argued?
     How can we counter this argument?

2. **Pipelining II**

   Write a benchmark application that allows to measure and compare throughput of scalar operations (addition, division, multiplication of integers (int, 4 bytes)) and floating point operations (double, 8 bytes) in the following two scenarios:

   Scenario I: No (or infrequent) dependencies between operations so that CPU can pipeline instructions effectively.

   Scenario II: Every operation depends on the result of a previous operation so that no pipelining is possible.

   Document and discuss your results:

   How does throughput change depending on the chosen operation (add/mul/div), data type (int/double), and scenario?
   ($3 \times 2 \times 2 = 12$ results)

   Submit your measurement results, your source code, and the Makefile used to build the benchmark application.

   Notes:

   - Read the README file attached to this assignment.
   - A Makefile and a C source template is provided to build your application.
     You may modify the Makefile or write your own, of course.

- Avoid caching of operation results as the benchmark would otherwise measure memory access instead of ALU performance.
- Implement the benchmark application in C, assembly and/or intrinsics are not required (but viable).
- Use compiler options like `-mno-sse` for gcc to avoid generating MMX/SSE/AVX instructions and to restrict measurements to conventional scalar operations (this is already configured in the attached Makefile).

3. Data- and Instruction Processing

We discussed several concepts of parallel processing in the lecture:

- What are the conceptual differences between pipelining, vector processing and multi-threading?