Chapter 1 : Software optimization for high-performance computing (edition) | Open Library

*Software Optimization for High Performance Computing: Creating Faster Applications [Kevin R. Wadleigh, Isom L. Crawford] on calendrierdelascience.com *FREE* shipping on qualifying offers. This is the most hands-on guide to writing high-performance software.*

Once you start asking questions, innocence is gone. There are four major components that determine the speed of an application: You do have control over your source code and how compilers are used on it. This book discusses how to perform source code modifications and use the compiler to generate better performing applications. The final and arguably the most important part is the algorithms used. By replacing the algorithms you have or were given with better performing ones, or even tweaking the existing ones, you can reap huge performance gains and perform problems that had previously been unachievable. There are many reasons to want applications to execute quickly. Sometimes it is the only way to make sure that a program finishes execution in a reasonable amount of time. For example, the decision to bid or no-bid an oil lease is often determined by whether a seismic image can be completed before the bid deadline. Since developers of applications would like an advantage over their competitors, speed can sometimes be the differentiator between two similar products. Thus, writing programs to run quickly can be a good investment. The individual tools are the various optimization techniques discussed. As expected, some tools are more useful than others. Reducing the memory requirements of an application is a general tool that frequently results in better single processor performance. Other tools, such as the techniques used to optimize a code for parallel execution, have a more limited scope. These tools are designed to help applications perform well on computer system components. You can apply them to existing code to improve performance or use them to design efficient code from scratch. As you become proficient with the tools, some general trends become apparent. All applications have a theoretical performance limit on any computer. The first attempts at optimization may involve choosing between basic compiler options. The next steps may involve more complicated compiler options, modifying a few lines of source code, or reformulating an algorithm. The theoretical peak performance is like the speed of light. As more and more energy, or time, is expended, the theoretical peak is approached, but never quite achieved. Before optimizing applications, it is prudent to consider how much time you can, or should, commit to optimization. In the past, one of the problems with tuning code was that even with a large investment of a time the optimizations quickly became outdated. For example, there were many applications that had been optimized for vector computers which subsequently had to be completely reoptimized for massively parallel computers. This sometimes took many person-years of effort. Since massively parallel computers never became plentiful, much of this effort had very short-term benefit. In the s, many computer companies either went bankrupt or were purchased by other companies as the cost of designing and manufacturing computers skyrocketed. For example, they nearly all have high-speed caches. Thus, making sure that code is structured to run well on cache-based systems ensures that the code runs well across almost all modern platforms. This is because they are most characteristic of modern high performance computing. They are written in Fortran, C, or in a language-independent pseudocode. Fortran examples use uppercase letters while the others use lowercase. The notation may also make it more difficult to compilers to optimize the source code. There is an entire chapter devoted to language specifics, but pseudo-code and Fortran examples assume that multidimensional arrays such as Y , have the data stored in memory in column-major order. Thus the elements of Y , are stored as Y 1,1 , Y 2,1 , Y 3,1 , This is the opposite of C data storage where data is stored in row-major order. Courier font will be used for all examples. Mathematical terms and equations use italic font.

Chapter 2 : Software Optimization for High Performance Computing: Creating Faster Applications | InformI

The hands-on guide to high-performance coding and algorithm optimization. This hands-on guide to software optimization introduces state-of-the-art solutions for every key aspect of software performance - both code-based and algorithm-based. Two leading HP software performance experts offer.

It is a hardware-level instruction that deals with circuit-based operations. Different units have specific functionalists. ALU is a digital circuit that perform integer arithmetic and logical operations. FPU contains status e. It also reads and interprets instructions. Bus Unit connects the major components of processor. It is a digital circuit than can shift a data word by a specified number of bits in one clock cycle. It specifies the op-codes Operation codes for various operations add, move, store, load, jump etc along with format and specification of operands. As a name suggest, each instruction in a code can execute several operations and can take multiple clock cycles i. Examples are Intel x86, Motorola 68k etc. These bits are converted into datapath signals by control units. CISC processor provides variety of addressing modes allowing constants and other operands either in memory or registers. So, the instruction size depends on the size of the operands. The general instruction formats for register-register and memory-memory addressing mode is showed in Fig. The feature enables instructions to be executed per cycle. CISC is motivated by supporting efficient access to complex data structures, providing flexibility to access operands. As an example, Pentium provides based-indexed addressing with scale factor to access complex data structures such as multi-dimensional array. General Instruction format for Intel Pentium is showed in Fig. If you are multiplying the data a and b that resides in the memory and store the result back to the memory as showed in Fig A7. In CISC architecture, you just need one instruction. It implicitly loads and stores acting as high level language statement into assembly. MULT a, b So, in CISC, the length of the code can be sufficiently smaller and compiler has to do a little work to translate the code into assembly and it requires less RAM to store instructions on the expense of more transistors of hardware for control logic. The separate load and store instruction in RISC also reduces the amount of work as the value of the operand in the register can be re-used for another operation rather than re-loading it. Big-endian system stores the most significant byte of a word in the smallest address where as Little-endian stores the least significant byte of a word as showed in Fig. Here, the word e. Endian Representation of data storage in a memory Memory Bank Fig. A9 is a logical unit of storage. It provides independent access to data â€" separate addresses and data lines. A quantity of memory that is wide enough to match the bit width of the data bus is called a bank of memory. In a single read or write operation, only one bank is accessed. Multi-bank Architecture â€" banks, rows and columns; the requested row is activated and is copied to the row buffer of the corresponding bank Many chips are combined on a memory module as showed in Fig. Data in 2X3 array 1 2 3; 4 5 6 is laid out contiguously as 1 2 3 4 5 6 and 1 4 2 5 3 6 in linear memory for row-major and column-major storage respectively. The adjacent element in the column for row-major are 3 words apart 2X3 dimension i. Accessing array elements that are contiguous in memory is usually faster due to caching. So, when adjacent addresses are accessed, each bank services them, resulting in good utilization. In two dimensional array elements a[0][0], a[1][0], a[2][0], â€¦ are all in the same memory bank. Instruction Pipelining Instruction pipelining allows overlapping execution of multiple instructions with the same circuitry in stages optimizing the instruction throughput. In the 5-stage pipeline as showed in Fig. Data Memory Access read , WB: Register Write Back Reducing data dependencies is the important part for optimization besides keeping the pipeline stages as busy as possible. The stages in original Pentium processors are prefetch, decode1, decode2, Execute, and Write Back as showed in Fig. The instruction is fetched from the cache to be ready for the next clock cycle. In this stage the instruction is decoded into opcodes and data. On every clock cycle, the decoder tries to decode three instructions that can happen only if the three instructions one complex and two simple instructions matches the capability of decoders i. So, here is the opportunity of optimization by re-arranging instructions. The unparable instructions NP and pairable UV instructions are categorized first.

The instructions issued on U-pipe are then paired with a suitable instruction in the V-pipe. The intel optimizing compiler can schedule code for both Pentium III and IV processors providing efficient instruction decoding. In this stage, the instructions ready to be executed are searched and executed in the fastest possible order employing available ports connected to execution units as showed in Fig. Dispatch ports and Execution Units in Pentium 4; clock x2 implies that it can execute two micro-instructions per clock cycle; CPU can execute upto 7 micro-instructions simultaneously port 0 and port 1 ; two double speed ALUs implies that Pentium 4 is optimized for simple operations. Port 0 is used during store operations to send the data to be stored at the address which is generated by either ALU or FPU depending on the data to be stored. Port 1 can execute simple instructions even when FPU is busy. The execution stage can therefore be optimized with the good blend of instructions which can keep all the ports busy through low dependency. Instruction latency is the number of clock cycles required to complete one instruction once the inputs are ready and execution begins. Instruction Throughput is the number of clocks cycles the processor is required to wait before starting the execution of identical instruction. Instruction Performance for Pentium 4 is tabulated in Table 1. Now, consider the C code fragment that has dependencies:

Chapter 3 : Software Optimization for High Performance Computing: Creating Faster Applications by Isom

The hands-on guide to high-performance coding and algorithm optimization. This hands-on guide to software optimization introduces state-of-the-art solutions for every key aspect of software performance both code-based and algorithm-based.

Chapter 4 : HPC: High-Performance Computing

Software Optimization for High Performance Computing has 1 rating and 0 reviews. This is the most hands-on guide to writing high-performance software. Us.

Chapter 5 : Code Optimization @HPC - High Performance Computing Cluster at CWRU

The performance at which the FP units generate results for multiply and add operations is measured inï¬‚oating-point operations per second (Flops/sec). The reason why more complicated arithmetic (divide, square root, trigonomet-.