LabVIEW Real Time for high performance control applications

Aljosa Vrancic, Lothar Wenzel
National Instruments
aljosa.vrancic@ni.com, lothar.wenzel@ni.com

Abstract

We show how LabVIEW Real-Time can be used to execute a 3k I/O point control loop in less than 1 ms on a single off-the-shelf 8-core workstation. At the heart of the control loop is an optimized 3k x 3k symmetric-matrix-vector multiplication algorithm. We report findings from ongoing work on a fast multiplication algorithm for 15k x 9k by 15k matrix-vector multiplication using a cluster of workstations. As a second study of applying LabVIEW to high performance real-time control we discuss a novel approach for solving general linear and non-linear elliptic partial differential equations (PDE). The result is that a non-linear PDE is solved in less than 1ms on a 111x55 grid for a tokomak fusion control application. For both studies we provide additional results obtained by implementing these new algorithms on GPUs.

Extremely Large Telescope

M1 (the primary mirror) is a segmented mirror consisting of 984 hexagonal mirrors (Figure 1a) each with a diameter in the 2 m range (for comparison: the Hubble Space Telescope's main mirror has a diameter of 2.4 m). While in operation, adjacent mirrors (Figure 1b) might be slightly tilted against each other, and this deviation from the ideal situation can be sensed by edge sensors. The spatial dimensions of M1 complicate things as both sensor and actuator data have to travel over longer distances deterministically. The goal is to maintain a perfectly aligned field of mirrors at all times with a loop-time of 1 ms. The model that emerges relies on a 3,000-by-3,000 matrix-vector multiplication where the symmetric matrix is

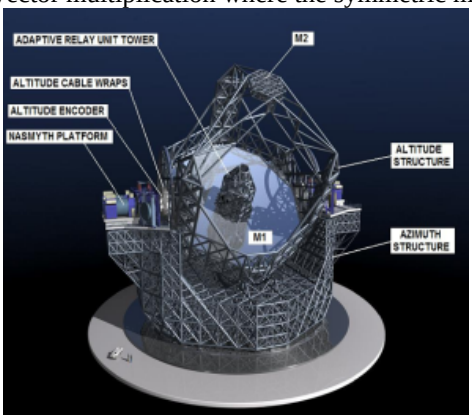

Figure 1a: ESO's Extremely Large Telescope

known in advance. ESO's M4 mirror is computationally more demanding. It is currently specified as a 2 m deformable mirror driven by 8,000 small actuators. The actuator data is generated by analyzing incoming wavefronts that deviate from the ideal because of atmospheric disturbances. Dense linear algebra of matrices and vectors of sizes 10k and beyond is required where the time-constraints are again in the sub 1 ms range. It is estimated that such real-time reconstruction algorithms can be handled efficiently using 10-20 blades with 8 cores each. We are currently benchmarking such algorithms.

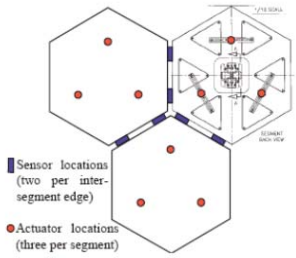

Figure 1b: Sensors between adjacent mirror segments

Algorithm

We identified that the required performance for M1 can be achieved only if the symmetric matrix can be kept in on-CPU cache at all times and the full power of SSE2 instruction is utilized. To do so, we repack the matrix data off-line to half of its original size and organize its contents in 4x4 blocks. This particular data organization utilizes all 8 SSE registers during computation. The packed matrix data is then distributed across the processors/cores in such a way that it stays inside on-CPU cache for the duration of the application's execution and that each CPU can solve a symmetric part of the problem. During the execution, one of the CPUs obtains new vector data, distributes it to the rest of the CPUs, and collecting results from the other CPUs back into a single output vector. To handle problem sizes that do not fit inside the cache, the algorithm must perform calculation in "both directions." This means that the algorithm is able to toggle the direction in which it accesses data. For example, let's assume that 10% of the data does not fit into the cache and that data is de-cached using LRU (least recently used) approach. If now application starts accessing data from the beginning, it will have to reload the first 10% of data from main memory causing the next 10% of the data that is already in memory to be decached. The process of decashing data that will be used next continues causing the algorithm to always run out-of-cache. On the other hand, if the algorithm is able to toggle the direction of calculation, all but the last 10% of data will already be in the cache so, only 10% of data will have to be accessed out of main memory. We observed speedups of up to 50% when employing the direction toggling approach.

Benchmark

All reported results were obtained using Dell 7400 workstation with two quad-core Intel Xeon processors running @ 2.6 GHz/12 MB cache per processor, 4 GB of RAM, and LabVIEW Real Time 8.5.1. For problem sizes

that fit in on-CPU cache the execution time should be proportional to the CPU clock frequency. For example, a 3.2 GHz version of the processors should reduce execution times in Table 1 by 20-25%. Unfortunately, the faster execution does not enable problem size increase because of the 24 MB on-CPU cache limit. As the new 30MB on-CPU cache quad-core processors are released, the problem size can be increased by 20-25% and still meet 1 ms timing constraint.

Vector/Matrix Size	Num CPUs	Time (μs) (average/best/worst)
3k x 3k	4	600/500/700
3k x 3k	8	500/450/650
3390 x 3390	4	worst case 950

Table 1: Benchmark results for symmetric matrix-vector multiplication

Nonlinear PDE

The primary motivation behind the new approach to solving (non)-linear PDEs has been nuclear fusion with its challenge of plasma control in tokamak confinements. The magnetic flux of the plasma for a fixed cross-section (R-radial component, Z-vertical component) of the toroidal tokamak container is described by the non-linear Grad-Shafranov partial differential equation.

Smaller coils sense the field resulting from the interaction between actuators and plasma. The measurements are used to reconstruct the magnetic flux by fitting PDE solutions of against those measurements. These solutions are used to drive the actuator coils to maintain the plasma shape. The cycle time must be on the order of 1 ms or faster. Our goal has been to replace the current PCA-based approximation for the Grad-Shafranov solver with a more accurate solver. The nonlinearities can be approximated by low-order polynomials in R and in the unknown function ψ.

Algorithm

We start with a general elliptic linear PDE specified by Dirichlet boundary conditions. Using finite difference approach, the equation is turned into a set of linear equations. For Neumann boundary conditions, Dirichlet/Neumann hybrid cases, or cases where the boundary conditions are given both at the edge and at the internal points, a slightly different set of equations emerges but the same approach applies.

Given the inverse matrix, the real-time part of the calculation consists of multiplying the inverse matrix with the boundary condition vector BC. A brute force approach for an NxN grid takes on the order of N^4 MAC (multiply and accumulate) operations to obtain the solution. For a 128x128 grid, the computation requires 268M operations. The calculation is limited by the memory-to-CPU bandwidth because the data set cannot fit in on-CPU cache. Assuming 10 GB/s memory-to-CPU throughput, each iteration takes more than 27 ms to calculate, well beyond the desired goal of 1 ms.

The first reduction in the number FLOPs required utilizes the system's dependency on roughly 4N boundary conditions. By manipulating columns the matrix model is reduced and the cost of the evaluation of all points on the NxN grid drops to $4N^3$, or 8.4M MAC steps for 128x128 grid. This corresponds to 17 GFLOPs for the calculation to meet 1 ms time limit. This is achievable if the system has at least 36 MB of cache (each point is 4 bytes). If not, the speed is again gated by the memory-to-CPU bandwidth producing calculations that take approximately 4ms, assuming a 10 GB/s bandwidth.

We use a novel approach to further reduce the number of required real-time calculations. A finite difference version of the PDE implies that a change in boundary conditions will propagate through the grid like waves until the grid finally settles to the solution. Only one path between two grid points is needed for changes in their values to affect each other. Figure 2 shows how can a grid be divided into two smaller independent sections by calculating solution on a subsection of the grid. Since the values of the calculated grid points do not change (they represent the partial solution), they must represent boundary conditions for the smaller two sub-grids. Smaller grids result in fewer calculations. The subdivision process is repeated until a threshold size grid is reached. Coefficients necessary to calculate solution on new boundary grid points can be calculated off-line.

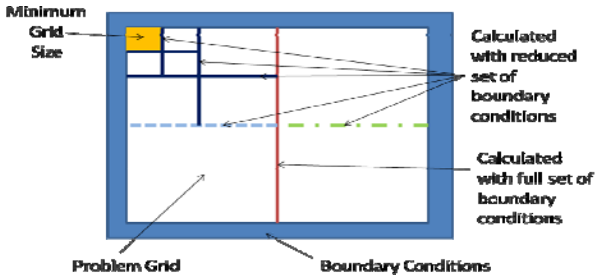

Figure 2: Dividing the solution grid to reduce overall number of calculations

The total number of MAC steps required for calculating solutions on a 127x127 grid size partitioned to the minimum sub-grid size of 7x7 as shown in Figure 2, is 792573, or 1.6 GFLOPs for execution computing in 1 ms. In general, the calculation cost scales for a 2D NxN grid as $O(N^2 log N)$. To handle larger grids, we developed a new way of calculating rows of inverse matrices required for calculation of grid points used for partitioning.

The described process can be directly applied to non-linear PDEs with few changes. The non-linear PDE is solved in an iterative fashion in which result for step *n* is used as a starting point for step *n+1*. The major difference with respect to the linear PDE algorithm is that the solution for each point now, in addition on the boundary conditions, also depends on the current values of the grid points. Consequently, the number of calculations required for solution on a grid of a given size increases.

GPUs

One can run either into limitations in CPU computational power or into bus bandwidth limitations if the problem data does not fit into the on-CPU cache. The memory bandwidth is one area where GPUs currently have lead over general purpose CPUs, so much so, that on GPUs one may treat all memory as cache. This is the main reason why we are investigating how to map algorithms described in this paper onto GPUs.