Field Precision LLC

Speeding Up

Stanley Humphries, Ph.D.

Field Precision LLC
E mail: techinfo@fieldp.com
Internet: https://www.fieldp.com

Our software handles two challenging classes of physical problems: finite-element solutions for electromagnetic fields and Monte Carlo solutions for electron/photon interactions in matter. Both types push the limits of modern personal computers, so speeding up solutions is a primary concern. The programs have three features to maximize the information output rate:

- Programs are coded in FORTRAN using optimized Intel compilers.
- They support parallel operation in multi-core computers.
- Multiple instances of programs may be launched in separate threads on multi-core computers, working on solutions simultaneously.
- Programs may be called from batch files to run in the background so the computer can address large solution sets autonomously while users perform other tasks.

This tutorial concentrates on the second and third features with the goal of defining realistic expectations for multi-core operation.

Our 3D electromagnetic codes support **OpenMP** for parallel operation within a single instance of a program. Here, the user has the option to specify the number of cores for the division of the task. In theory, a calculation employing four cores should run in one fourth the time of a non-parallel solution. In practice, the speed advantage for finite-element calculations is significantly less for several reasons:

- Only one process at a time can access a file in sequential disk operations. In programs that involve the creation of large files, data transfer can be a significant bottleneck.
- Individual processes cannot simultaneously access the same memory location. In a large matrix inversion, the mesh must be divided into segments with special precautions at common boundaries. Depending on the mesh size, there is a point where the division becomes counter-productive, limiting the number of cores.
- Organizing the task division adds computational overhead.
- **OpenMP** operations may conflict with acceleration processes built in to the CPU.

As an illustration, I made tests of **OpenMP** in a practical calculation with **MagNum**, our 3D magnetic field code. Such a solution includes an initial extended disk transfer to generate the coefficients for the finite-element matrix inversion. I compared the most recent versions of **OpenMP** and the Intel compiler to versions from a decade ago and found no significant difference in the run times. I ran tests on two computers, a Windows 7 machine with an Intel Core i7 chip (clock speed 2.93 GHz) and a Windows 11 machine with a Core i3 chip at 3.30 GHz. Figure 1 plots the run time versus the number of threads. There are some interesting features:

- With a single thread, the inexpensive Core i3 machine outperformed the old workstation by almost a factor of 3, significantly higher than the ratio of clock speeds. The implications are 1) there has been progress in chip design in the last decade and 2) the Core i3 machine is doing a lot of load optimization on its own.
- The use of two threads on the Core i7 machine reduces the run time by about 2/3. There is only marginal improvement with more threads. Even though the Windows Task Manager shows the CPU has gone from 12.5% to 100%, the maximum run time reduction factor is only 1.67. This results partly from the single-thread disk operations which occupy 78 seconds of run time and partly the computational work to implement multi-processor operation.
- On the Core i3 machine, there is a moderate run time reduction factor of 1.24 with two threads, but then the run time actually increases with the number of threads. With the maximum of 6 threads, the run time is reduced by only about 10%, even though the CPU is running flat out. This may indicate that the **OpenMP** processes conflict with optimization processes built into the chip.

An alternative to running one calculation in multiple threads is to run multiple calculations, each with its own thread. The simplest option is to run multiple instances of a program or several programs at a time. Each instance occupies its own space in memory, so there is no conflict. The one precaution is that the instances should not simultaneously access the same file. As a test, I launched **Magnum** three times in different windows to perform calculations in different directories. On the Core i7 machine, the solution time when running a single instance of the program was 351 seconds. Launching three independent solutions gave run times of 517, 526 and 532 seconds. I confirmed that the CPU usage increased from 12.5% to 38%. The longer time for each run was due partly to simultaneous disk operations during generation of finite-element matrix coefficients. Nonetheless, the procedure generated three times as much data, so the average time per data set was 175 seconds. The use of 3 cores use raised the data generation rate by a factor of 2.0. Results were less favorable on the Core i3 machine. I doubled the number of matrix inversion steps to maintain similar solution times. The

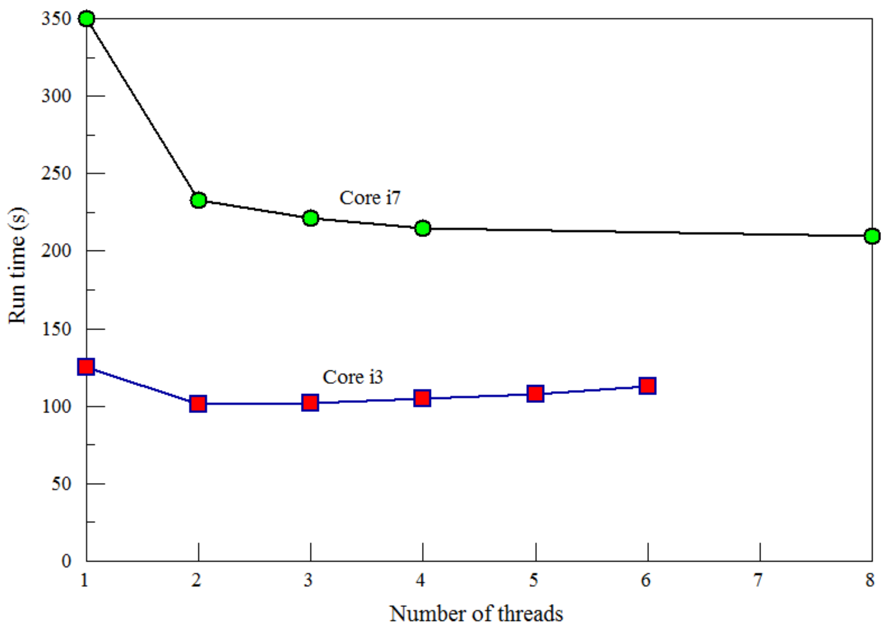

Figure 1: **OpenMP** test: run time of the benchmark example as a function of the number of threads for two computer configurations.

run time for a single instance was 220 seconds. Three simultaneous solutions had average run times of 456 seconds. The data generation rate was raised only by a factor of 1.4. As with the **OpenMP** results, I suspect there are deep optimization processes occuring in the newer chip so the threads do not operate independently.

Another way to utilize multiple cores is by launching multiple calculations in the background. All Field Precision programs can run as background tasks called from the Windows command prompt or batch files. I made a test setup with mesh and winding files in a directory C:\simulations and three instances of the **Magnum** input file in subdirectories. A batch file in the directory has the content:

```
START /B C:\fieldp/amaze/magnum.exe C:\simulations\PC01\latching_solenoid
START /B C:\fieldp/amaze/magnum.exe C:\simulations\PC02\latching_solenoid
START /B C:\fieldp/amaze/magnum.exe C:\simulations\PC03\latching_solenoid
```

The relative run times and data rate advantages for the Core i7 and Core i3 machines were not significantly different from those obtained with instances launched in individual windows. In summary, parallel operation with **OpenMP** may help increase the data generation rate but advantages depend on the chip logic and disk access rate. With N_c cores, the data rate advan-

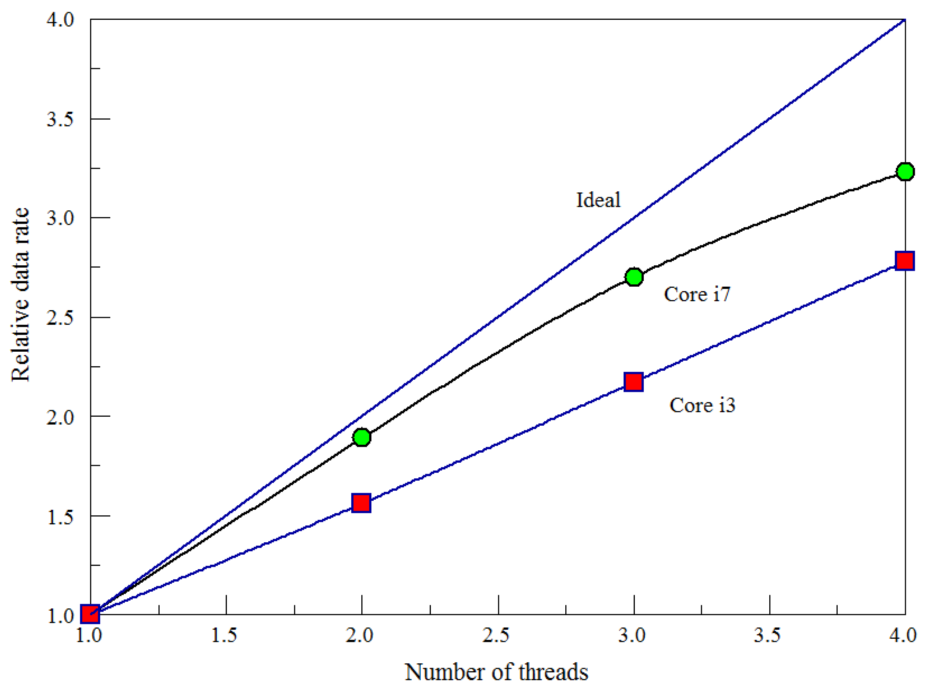

Figure 2: Relative data rates for **GamBet** calculations as a function of the number of threads for two computer configurations.

tage may be significantly less than a factor of N_c even though the Windows Task Manager shows that the CPU is doing N_c times as much work.

There is one exception where the data rate enhancement approaches N_c. Our **GamBet** program uses Monte Carlo techniques to model X-ray physics and electron interactions with matter. These calculations involve tracking showers for N_p input particles. This number must be large because the statistical accuracy scales as $\sqrt{N_p}$. Because primary and secondary particles move at random through the mesh, they must be tracked sequentially to avoid memory access conflicts. Therefore, parallelization with **OpenMP** is not possible. We developed our own approach for extensive runs. The control instance of the program runs in a window or in the background. It analyzes the input script, calculates material data and records the result in a file. The control instance then launches $(N_c - 1)$ instances of a background version of itself. The subtasks have different random number seeds so that the results are statistically independent. Each instance uses its own section of memory and creates independent files of dose distributions and particle escape statistics. When all calculations are complete, the control instance combines the data files to obtain results with enhanced statistical accuracy. File transfers are quick and the instances are offset in time, so there is no disk access conflict. I made test calculations on the two computers with different levels of parallelization. I kept N_p, the total number of shower calculations, the same and checked the advantage of dividing the work into multiple tasks. Figure 2 plots the results. The relative data rate equals the ratio of time to complete N_p showers with a single core divided by the multiple-core times. The data rate for the Core i7 machine is close to ideal. Even the Core i3 machine shows a significant improvement with multiple threads. For a comparison of the absolute data rate, note that the single-thread run time for the Core i3 machine is only one-third that for the older Core i7 machine.

The emphasis in this tutorial has been on maximizing the data generation rate using the multiple core capabilities of personal computers. Nonetheless, it's important to remember that there are time-saving methods that do not depend on the computer and are often more effective. Here are some RI techniques that can be major time savers in finite-element calculations:

- Make preliminary analytic estimates to avoid unproductive regions of parameter space in the computer calculations.
- Design meshes to avoid high element density and conformal shaping in regions that are not critical to the solution.
- Apply symmetry conditions and special boundaries to minimize the mesh size.
- Monitor numerical results as they become available to guide subsequent calculations.
- Make full use of advanced program features.

Regarding the last item, our 3D electric and magnetic field programs support a procedure for microscopic solutions. Here, a large scale solution provides boundary conditions to determine field distributions around extremely small features. The two-step approach is faster and more accurate than a single solution requiring a complex mesh to represent the large difference in scale.