



Integrator Column Quality Magazine

Date: 6-8-2009

Author: Ned Lecky

Lecky Integration

ned@lecky.com

Work/Cell: 518-258-5874

High Performance Custom Applications using OpenMP

I help people implement high-speed machine vision and automation applications. Many machine vision applications can be handled today with “smart camera” products. These all-in-one vision systems contain both camera and image processor, and package in carefully-designed software that allows a point-and-click configuration of entire machine vision applications- as long as your application is a standard one, like simple part presence/absence, reading a 1- or 2-D barcode, measuring a feature that is always in the same plane, or finding a simple pattern or template.

When an application calls for more customization, we move to one of several industrial machine vision application generators, such as VisionPro from Cognex, VPM/CPM from PPT Vision, or Sherlock from DALSA. These are large applications which combine a graphical setup and runtime environment intended to allow much more complex applications to be tackled without doing much, if any traditional C, C++, or Java coding. Such software includes image capture, image analysis with any of hundreds of algorithms, and I/O interfacing with the rest of a system through digital, analog, or network connections. The applications are at this point very mature and flexible, and if you are developing machine vision applications and you haven't seen all of them yet, you should take a look. They are very powerful.

But do they cover all applications? No. There are many highly-integrated machine vision applications that exceed the capabilities of the best automatic application generators. Consider the following example.

An Example

We have one camera waiting for electrical connectors coming down a conveyor belt. The connector pins are always visible, but may be in many different orientations. When a connector appears, we adjust three different light sources to get a solid image without blooming, read a barcode on a carrier, query a remote Oracle database that contains the expected pin locations, find four calibration features on the connector to establish a

measurement calibration plane, find the tips of all connector pins and measure their relative location to the plane, decide if the pins are within spec, control gating and part reject solenoids to eliminate failed parts, and post TIFF files to the database showing failed pins and their nearest neighbors. Do this at up four pieces per second using a \$2000 PC and a \$1000 GigE vision camera.

Why does this one need customization? The point-and-click programming environments actually have most of these capabilities, but the fine level of integration and control, and the need for management of remote I/O to a database, as well as a very custom-sounding

camera calibration process, contribute to my suspicion. The application is actually “easy” from a purely technical standpoint: each part is a relatively standard operation and a good software person should have no problem getting it working. The image-processing operations are all provided in the open source OpenCV library I discussed in my last column. But we will be on our own at the integration level trying to get everything to hum along at speed.

Going fast on a Multi-core PC

You’re probably aware of the latest trend in Intel and AMD processors for PCs: multicore technology. Technically, these processors implement what is called Shared Memory Parallel Processing: Each “processor chip” actually contains two or four processing cores, each with their own cache memories for acceleration, but all accessing the same main computer memory. For my \$2000 PC budget, for example, I can buy a nice Dell quadcore Intel server-grade machine. More expensive machines will provide multiple quadcore processors to provide even more crunching capability.

Great- but how to use the four cores? The onus is on the programmer to divide the application into at least four equally time-consuming processes or threads- we must decide what each core will be doing and how to keep it busy. This is called multithreaded programming, or parallel programming, and it is, well, a pain in the neck. It is a source of many of the difficult bugs that creep into modern software. It also is a huge detriment to code readability. By breaking a program into several parts that run simultaneously, the operation of the software as a complete solution becomes very obscure. And when things “almost” work, it may be entirely unclear what thread or task is the culprit.

Enter OpenMP (Open Multi Processing), my new best open source friend. I have recently been coding multi-processor applications using this set of compiler extensions developed by the open source computer science community. It is supported by many compilers, including the professional version of Visual Studio and follows a model in which you create your application as only a single thread. We would write and debug our test application in a completely sequential form first, and then add parallelism as desired later on using directives known as pragmas that merely direct the compiler to automatically generate and control additional threads.

For example, say the following pin location comparison step takes 100mS to run as is, since ComparePinLocations is a very complicated function that does lots of data reduction, calibration, and database comparison:

```
for(int i=0; i<numPins; i++)  
r[i]=ComparePinLocation(i,x[i],y[i],z[i]);
```

With OpenMP, we just add a line above the loop as follows:

```
#pragma omp for shared(numPins,x,y,z,r) private(i)  
for(int i=0; i<numPins; i++)  
r=ComparePinLocation(i,x[i],y[i],z[i]);
```

Compiling this new “version” with OpenMP enabled in Visual Studio, the execution time drop from 100mS to 51mS with a dual-core machine, and to about 27mS with a quadcore machine. Magic? No- the compiler just automatically generated multiple threads, dividing up the for loop into sections and assigning them to different processors.



As someone who has hand-crafted many parallel processing programs, I can assure you that the advantage of keeping things this straightforward is enormous. And if something stops working in the future, I can just turn off OpenMP support and recompile, and I'll be back to a single thread version for testing which will let me see if my issues are related to multithreading or to something more basic.

OpenMP handles not just loop rethreading, but also parallelizing entire sections of code that you know can be run in parallel. In the example, these are the database lookup, update, image acquisition and solenoid control sections. I came very close to quadrupling the speed of my application by adding a total of 30 lines of pragmas to my sequential program!

For sure, becoming an expert with OpenMP requires some practice and experimentation. The OpenMP.org website and the text "Learning OpenMP" from MIT Press are very helpful and loaded with real, practical examples, for both the C and the FORTRAN programmer. So check them out- you may suddenly be writing optimized multithreaded programs to take advantage of multi-core processors! Put it on your resume.