GammaLib - Feature #226

Parallelize maximum likelihood computation

06/16/2012 12:50 AM - Knödlseder Jürgen

Status:	Closed	**Start date:**	06/16/2012
Priority:	Normal	**Due date:**	
Assigned To:		**% Done:**	100%
Category:		**Estimated time:**	175.00 hours
Target version:	Stage Jean-Baptiste Cayrou		

Description

Exploit the availability of multiple cores for the maximum likelihood computation. Parallelization can be implemented in the observation loop or in the event loop (the observation loop is likely easier to implement).

This needs some control of the number of cores that are available for computation. There are several options for handling this:

- information management by a global class
- number of cores set using an environment variable
- number of cores set at compile time (not flexible at run time)

Subtasks:

Action # 261: Study possible options for parallelization	**Closed**
Action # 263: Validate the test program	**Closed**
Action # 262: Create test program for code parallelization	**Closed**
Action # 265: Validate parallelization of GammaLib	**Closed**
Action # 264: Implement code parallelization in GammaLib	**Closed**
Action # 399: Create documentation about parallelization	**Closed**

History

#1 - 06/24/2012 03:36 PM - Knödlseder Jürgen

Here some information gathered from the Web.

http://stackoverflow.com/questions/2166425/how-to-structure-a-c-application-to-use-a-multicore-processor
Basically, you need to multithread your application. Each thread of execution can only saturate one core. Separate threads tend to be run on separate cores. If you are insistent that each thread ALWAYS execute on a specific core, then each operating system has its own way of specifying this (affinity masks & such)... but I wouldn't recommend it.
Like what Pestilence said, you just need your app to be multithreaded.

http://apais.developpez.com/tutoriels/c++/multithread/

http://www.devarticles.com/c/a/Cplusplus/Multithreading-in-C/
By not building in support for multithreading, C++ does not attempt to define a "one size fits all" solution. Instead, C++ allows you to directly utilize the multithreading features provided by the operating system. This approach means that your programs can be multithreaded in the most efficient means supported by the execution environment. Because many multitasking environments offer rich support for multithreading, being able to access that support is crucial to the creation of high-performance, multithreaded programs.

http://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm

http://www.linuxdocs.org/HOWTOs/C++Programming-HOWTO-24.html
Show an example of a C++ thread class. Not sure that we exactly need this. Our goal is to execute class methods as threads. See next link.

http://stackoverflow.com/questions/4666635/run-threads-of-class-member-function-in-c
Use std::thread

http://en.cppreference.com/w/cpp/thread/thread

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2184.html

#2 - 06/24/2012 08:59 PM - Knödlseder Jürgen

And here some requirements:

- make sure that there is no problem with concurrent memory access
- parallelization should also work under Python (watch for problems with GIL - Global Interpreter Lock)
- parallelization should be user configurable

#3 - 06/24/2012 09:01 PM - Knödlseder Jürgen

And here a possible workplan:

- Study possible options (thread, fork, etc.)
- Create a test program
- Validate parallelization with test program (verify fulfillment of requirements)
- Implement in GammaLib
- Validate in GammaLib

#4 - 07/02/2012 12:02 PM - Knödlseder Jürgen

- Target version set to Stage Jean-Baptiste Cayrou

#5 - 07/28/2012 12:53 AM - Knödlseder Jürgen

- Target version deleted (Stage Jean-Baptiste Cayrou)

#6 - 07/28/2012 12:53 AM - Knödlseder Jürgen

- Target version set to Stage Jean-Baptiste Cayrou

#7 - 07/30/2012 03:47 PM - Anonymous

- Status changed from New to In Progress

#8 - 09/01/2012 03:42 AM - Knödlseder Jürgen

- Status changed from In Progress to Closed