Vorlesung 8

Allgemeine Relativitätstheorie mit dem Computer

PC-Pool Raum 01.120 Johann Wolfgang Goethe Universität 06. Juni, 2016

Matthias Hanauske

Frankfurt Institute for Advanced Studies
Johann Wolfgang Goethe Universität
Institut für Theoretische Physik
Arbeitsgruppe Relativistische Astrophysik
D-60438 Frankfurt am Main
Germany

Allgemeines

Ort und Zeit:

PC-Pool Raum 01.120, immer Montags von 16.15 bis 17.45 Uhr
Zusätzlicher, freiwilliger Übungstermin 15.00 bis 16.15 Uhr

Vorlesungs-Materialien und *Lon Capa* Online-Lernplatform:

http://th.physik.uni-frankfurt.de/~hanauske/VARTC/

http://lon-capa.server.uni-frankfurt.de/

Plan für die heutige Vorlesung:

Das parallele C++ Programm zum berechnen der Tolman-Oppenheimer-Volkoff (TOV) Gleichungen einer Sequenz von Neutronen/Quark Sternen

1) Die OpenMP- C++ Version mit geordneter Ausgabe in eine Datei, variabler Zustandsgleichung und Terminalausgabe der benötigten Zeit
2) Die MPI- C++ Version mit geordneter Ausgabe in eine Datei, variabler Zustandsgleichung und Terminalausgabe der benötigten Zeit

Struktur und Performance der beiden Programme

Einführung: Rotierende Schwarze Löcher mit Maple

OpenMP

The **parallel computer language “OpenMP** (Open Multi-Processing)” supports multi-platform shared-memory parallel programming in C/C++ and Fortran. It is a collaborative developed parallel language which has its origin in 1997.

OpenMP separates the parallizable part of the program into several **'Threads'** where each thread can be executed on a different processing element (CPU) using shared memory.

OpenMP has the advantage that common **sequential codes can easily be changed** by simply adding some OpenMP directives. Another feature of OpenMP is that the program runs also properly (but then sequentially, using only one thread) even if the compiler does not know OpenMP.

Performance

The **performance of a parallel computer code** can be measured using the following characteristic values:

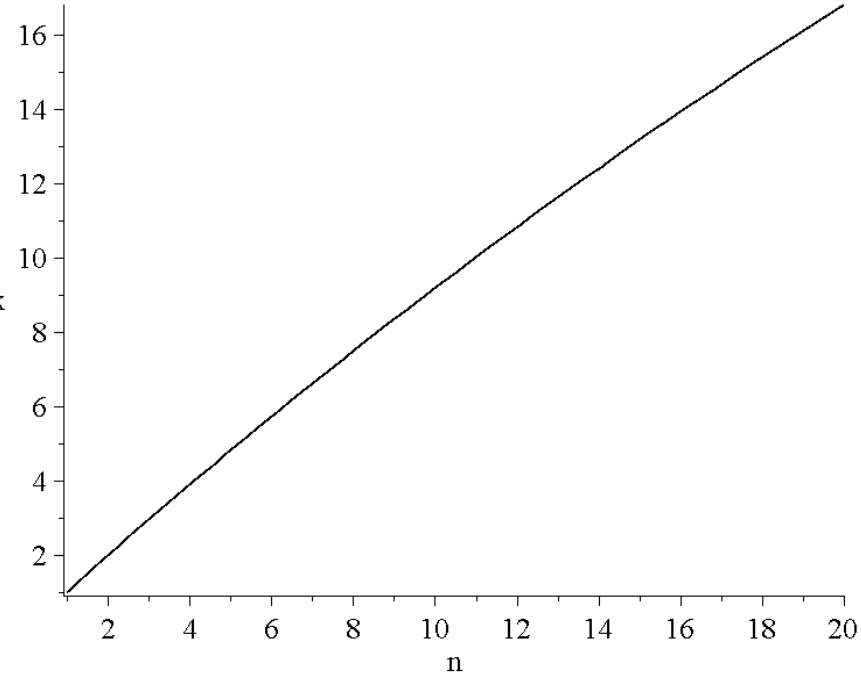
T(n): Time needed to run the programm on n processing elements (e.g. CPU's, computer nodes).

Speedup: S(n):=T(1)/T(n) , **Efficiency**: E(n):=S(n)/n

Amdahl's law:

The “Amdahl's law” describes the speedup of an optimal parallel computer code. T(n) is diveded in two parts (T(n)=Ts+Tp(n)), where Ts is the time needed for the non-parallizable part of the programm and Tp(n) is the parallizable part, which can be executed concurrently by different processors. A(n):=Max(S(n))

A(n)=1/(a+(1-a)/n), with a:=Ts/T(1)



Amdahl's law with a=[0.01,0.4]

Further OpenMP directives

Loop parallelization(#pragma omp parallel for ...):

- **Access to variables (shared(), private(), firstprivate(), reduction())**
- **Syncronisation (atomic, critical)**
- **Locking (omp_init_lock(),omp_destroy_lock(),omp_set_lock(),_unset_lock())**
- **Barriers (#pragma omp barrier)**
- **Conditional parallelization (if(...))**
- **Number of threads (omp_set_num_threads())**
- **Loop workflow (shedule())**

Additional material:

The OpenMP® API specification for parallel programming: http://openmp.org/
The Community of OpenMP: http://www.compunity.org/
OpenMP-Tutorial: https://computing.llnl.gov/tutorials/openMP/
Book: Parallel Programming in C with MPI and OpenMP, by Michael J. Quinn.
Book: Patterns for Parallel Programming, by Timothy G. Mattson, et.al.
Book: Using OpenMP, by Chapman,et.al.
Book: OpenMP by Hoffmann, Lienhart
Tutorium Examples: http://fias.uni-frankfurt.de/~hanauske/new/parallel/openmp/

MPI

The **parallel computer language "MPI** (Message Passing Interface)" supports multi-platform shared- amd distributed-memory parallel programming in C/C++ and Fortran. The MPI standard was firstly presented at the "Supercomputing '93"-conference in 1993.

With MPI, the whole computation problem is separated in different Tasks (processes). Each process can run on a different computer nodes within a computer cluster. In contrast to OpenMP, MPI is designed to run on distributed-memory parallel computers.

As each process has its own memory, the result of the whole computation problem has to be combined by using both point-to-point and collective communication between the processes.

Further MPI Functions

Point-to-point message-passing:

- **Int MPI-Send(buff, count, MPI_type, dest, tag)** **e.g.**

```
MPI::COMM_WORLD.Send( &ergebnis[id], 1, MPI::DOUBLE, 0, 1);
```

- **Int MPI-Recv(buff, count, MPI_type, source, tag, stat)** **e.g.**

```
MPI::COMM_WORLD.Recv( &ergebnis[q], 1, MPI::DOUBLE, q, 1, status );
```

- **Collective Communication: MPI_Bcast**
- **Barriers: MPI_Barrier**
- **....**

Additional material:

MPI-Tutorial: https://computing.llnl.gov/tutorials/mpi/

Book: Parallel Programming in C with MPI and OpenMP, by Michael J. Quinn.

Book: Patterns for Parallel Programming, by Timothy G. Mattson, et.al.

MPI-Examples: http://people.sc.fsu.edu/~jburkardt/cpp_src/mpi/mpi.html

Tutorium Examples: http://fias.uni-frankfurt.de/~hanauske/new/parallel/mpi/