Fast Multipole Algorithm

Csilla Bükki, Edibe Kalelioglu
for a Programming Practice in main study period
at the University Bonn
led by Professor Andreas Weber

Summersemester 2004

Contents

1 Introduction

The Fast Multipole Algorithm (FMA) is an algorithm for the numerical solution of the N-body problem. Simply put, the classical N-Body problem involves computing the net effect of the interactions of each pair of particles out of a set of N. In molecular dynamics (MD) work, the particles are atoms and the forces are electrostatic. In astrophysical simulations, the particles are stellar bodies and the forces are gravitational. In either case the amount of computation grows as the square of the number of particles, for the naive implementation.

The FMA process, however, uses a Multipole Expansion (MPE) to represent the effects of a group of distant particles as a single entity. By using the MPE when computing forces on a particle, and doing operations to combine multipole expansions, the overall amount of computation can be reduced to an almost linear relationship with the number of particles.

Section 2 describes how we evaluated the dpmta package in VC.Net. We made it in C++ code, (it can be made in c code too). We have written additional some testprogramms in c code (it can be made in C++ code too). Section 3 shows that the dpmta package with our testprogramms is also executable in cygwin. In section 4 you can read about the testprogramms (written by us), which define different structure of the particles. The running time is various depending on the structure.

2 VC.Net

2.1 General

Under the website http://www.sentient.de/FMM/ppp.html you can find already executable programms for cpp and c application. In the cpp-application there are some testprogramms of the package DPMTA and in the c-application there are some testprogramms (written by us) which define different structure of the particles.

The following steps must be done to create the applications by self under Operation System Windows.

2.1.1 to make a new cpp-Application

At first, a new Project (Managed C++ Application) must be opened in VC.Net. There is a main cpp-file named as the project. The content of this file must be deleted and the content of FastMultipole1.cpp must be copied into this file. These files are in additional.zip. There is directory named as the project. The directories of DPMTA package : comm, mpole, src, test must be copied in this directory. The content of additional.zip must be copied such a way:

```
fftest.h          >>>    mpole
mpe_testC.h       >>>    mpole
mpe_testLJ.h      >>>    mpole
dpmta_test.h      >>>    test
```

In the directories "Source Files" and "Header Files" of the project the following new folders must be added: comm, mpole, src, test.
About FastMultipole1.cpp : argc is the number of the testprogramms which is calling by FastMultipole1.cpp.
For all c-files a new cpp -file should be created with the same name. The content of the c-file must be copied in cpp- file.
All cpp-files must be contain the following pseudo-code , (which is found in Fast-Multipole1.cpp too).

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;
```

In section Particular can be seen more of changes of source code, which must be done. "Ln" means the line of source code and the number shows circa the location of the change, where it can be found.
The cpp and header files, which can be seen in the Table (in Appendix) must be added to the project.
After these steps the whole projekt must be build (compile as C++ code and link). The following test file is executable (only local):

```
FastMultipole1.exe
```

The FastMultipole1.exe has a callnumber : each number belongs to a testprogramm.

```
1 ffttest
2 mpe_testC
3 mpe_testLJ
4 dpmta_test
```

2.1.2 to make a new c-Application

In this application cpp-files are not needed. There is directory named as the project. The directories of DPMTA package : mpole, src, test must be copied in this directory. The content of additional.zip must be copied such a way:

```
C2.c                                    >>>    test
no_pvm_mpi.c    &       no_pvm_mpi.h    >>>    test
line.c          &       line.h          >>>    test
lineeps.c       &       lineeps.h       >>>    test
grid2D.c        &       grid2D.h        >>>    test
grid3D.c        &       grid3D.h        >>>    test
star2D.c        &       star2D.h        >>>    test
star3D.c        &       star3D.h        >>>    test
```

The c and header files, which can be seen in the Table (in Appendix) must be added to the project in the directories : mpole, src,test. All cpp-files and stdafx.h must be deleted from the project. C2.c must be copied in the directory : "test" and added to the project (SourceFiles/test).
In section Particular can be seen the changes of source code , which must be done. After these steps the whole project must be build (compile as C code and link) (dont use precompile header for c-files!).
The following test file is executable (only local):

```
C2.exe
```

The C2 has a callnumber : each number belongs to a testprogramm.

```
1 no_pvm_mpi
2 line
3 lineeps
4 grid2D
5 star2D
6 grid3D
7 star3D
```

2.2 Particular:

Changes of the source code, which must be done.

2.2.1 comm

<div align="center">

comm_prims.cpp/c

</div>

Header stdlib.h must be included

2.2.2 mpole

<div align="center">

mpe_fft.cpp/c

</div>

Header stdlib.h must be included.

<div align="center">

ffttest.cpp/c

</div>

- Header ffttest.h must be included.

- Header time.h must be included.

- Ln 29 instead of

  ```
  main(int argc, char **argv)
  ```

  must be pasted

  ```
  ffttest_main(char **argv)
  ```

- Ln 51 instead of

  ```
   p = atoi(argv[1]);
   np = atoi(argv[2]);
  ```

  must be pasted

  ```
   p = 20;
   np = atoi(argv[3]);
  ```

- Ln 46 must be deleted

  ```
  if ( argc != 3 ) {
      fprintf(stderr,"Usage: %s <mp> <nparts>\n",argv[0]);
      exit(-1);}
  ```

- Ln 81 must be added

  ```
   /* Seed the random-number generator with current time so that
     * the numbers will be different every time we run.
     */
    srand( (unsigned)time( NULL ) );
  ```

  and instead of

```
  for ( i=0; i<np; i++) {
     vp1[i].x = drand48() - 0.5;
     vp1[i].y = drand48() - 0.5;
     vp1[i].z = drand48() - 0.5;
     q1[i] = 10.0 * (Real)(((i%2)*2)-1);
   }
```

must be pasted

```
  for ( i=0; i<np; i++) {
     vp1[i].x = (Real)rand()/(Real)RAND_MAX - 0.5;
printf("x1 : %6g\t" ,  vp1[i].x );
     vp1[i].y =  (Real)rand()/(Real)RAND_MAX - 0.5;
     printf("y1 : %6g\t" ,  vp1[i].y );
     vp1[i].z =  (Real)rand()/(Real)RAND_MAX - 0.5;
     q1[i] = 10.0 * (Real)(((i%2)*2)-1);
     printf("z1 : %6g\n" ,  vp1[i].z );
   }
```

- instead of

```
  for ( i=0; i<np; i++) {
     vp2[i].x = drand48() - 0.5;
     vp2[i].y = drand48() - 0.5;
     vp2[i].z = drand48() - 0.5;
     q2[i] = 10 * (Real)(((i%2)*2)-1);
   }
```

must be pasted

```
  for ( i=0; i<np; i++) {
     vp2[i].x = (Real)rand()/(Real)RAND_MAX  - 0.5;
     printf("x2 : %6g\t" ,  vp2[i].x );
vp2[i].y =  (Real)rand()/(Real)RAND_MAX - 0.5;
     printf("y2 : %6g\t" ,  vp2[i].y );
     vp2[i].z =  (Real)rand()/(Real)RAND_MAX - 0.5;
     printf("z2 : %6g\n" ,  vp2[i].z );
     q2[i] = 10 * (Real)(((i%2)*2)-1);
   }
```

mpe_testC.cpp

- Header mpe_testC.h must be included

- Ln 26 instead of

```
main(int argc, char **argv)
```

must be pasted

```
mpe_testC_main(char **argv)
```

mpe_testLJ.cpp

- Header mpe_tetsLJ.h must be included

- Ln 25 instead of

  ```
  main(int argc, char **argv)
  ```

  must be pasted

  ```
  mpe_testLJ_main(char **argv)
  ```

mpe_mpoleC.cpp/c

Ln 710 instead of

```
alpha = M_PI_2;
```

must be pasted

```
alpha = 3.14/2.0;
```

2.2.3 src

dpmta_config.h   [cpp,c]

SERIAL flag must be defined.

dpmta_serial.cpp/c

Ln 58 function void Sort_Particles() of dpmta_serial.cpp takes 2 parameters: (int, PmtaParticlePtr) (see at definition Ln 431)

dpmta_slvcompute.cpp/c

- Ln 444 function void Dist_Delete() of dpmta_discmisc.h/cp does not take 1 parameter: (Dpmta_NumLevels)

- Ln 446 function void Partition_Delete() of dpmta_distpart.h/cpp does not take 2 parameters: (Dpmta_NumLevels, Dpmta_Nproc)

dpmta_slvmkiil.cpp/c

Ln 127 instead of

```
#ifndef SERIAL
#include "comm.h"
#include "dpmta_message.h"
#endif
```

must be pasted

```
#include "comm.h"
#include "dpmta_message.h"
```

dpmta_timer.cpp/c

- Ln 175-176 in function void Init_Times:
  instead of

```
   {/* get clock ticks setting */}
   myclkticks = (double)sysconf(_SC_CLK_TCK);
```

must be pasted

```
 /* get clock ticks setting */

   #if defined WIN32
   myclkticks = (double)CLOCKS_PER_SEC;

   #else
   myclkticks = (double)sysconf(_SC_CLK_TCK);

   #endif
```

- Ln 251 in function void Start_E_Time must be deleted

```
gettimeofday(&runstruct,0);
```

- Ln 268 in function void Store_E_Time must be deleted

```
gettimeofday(&runstruct,0);
```

dpmta_timer.h   [cpp,c]

It has to be defined

```
typedef struct timeval {
    long    tv_sec;          // seconds
    long    tv_usec;         // and microseconds
    } TimeVal;
```

dpmta_slviter.cpp/c

- In Ln 759 instead of

```
void Delete_Local_Buffers();
```

  must be pasted

```
void Delete_Local_Buffers1();
```

- In Ln 740 instead of

```
void Init_Local_Buffers();
```

  must be pasted

```
void Init_Local_Buffers1();
```

dpmta_slviter.h   [cpp,(c)]

- In Ln 56 instead of

```
void Delete_Local_Buffers();
```

must be pasted

```
void Delete_Local_Buffers1();
```

- In Ln 57 instead of

```
void Init_Local_Buffers();
```

must be pasted

```
void Init_Local_Buffers1();
```

2.2.4 test

dpmta_test.cpp/c

- Ln 50 instead of

```
int main( int argc, char *argv[] )
```

must be pasted

```
int dpmta_test_main( char *argv[] )
```

- Ln 141 instead of

```
/* check and load command line arguments */
if (argc != 9) {
        fprintf(stderr,
            "%s <#procs> <#lvls> <#parts> <fft> <mp> <theta> <iter> <pbc>\n",
            argv[0]);
```

must be pasted

```
/* check and load command line arguments */
    if (9 != 9) {
        fprintf(stderr,
            "%s <#procs> <#lvls> <#parts> <fft> <mp> <theta> <iter> <pbc>\n",
            argv[0]);
```

- Ln 152 instead of

```
nprocs = atoi(argv[1]);
```

must be pasted

```
nprocs = atoi(argv[2]);
```

- Ln 155 intead of

```
initdata.nlevels = atoi(argv[2]);
    num_parts = atoi(argv[3]);
    initdata.fft = atoi(argv[4]);
    initdata.mp = atoi(argv[5]);
    initdata.theta = atof(argv[6]);
    iter = atoi(argv[7]);
    initdata.pbc = atoi(argv[8]);
```

must be pasted

```
initdata.nlevels = atoi(argv[3]);
    num_parts = atoi(argv[4]);
    initdata.fft = atoi(argv[5]);
    initdata.mp = atoi(argv[6]);
    initdata.theta = atof(argv[7]);
    iter = atoi(argv[8]);
    initdata.pbc = atoi(argv[9]);
```

3 Cygwin

The Package DPMTA can be used with cygwin. The executable testprogram can be found in directory "test" : dpmta_test.c.

We have written also testprograms, which don't use Parallel Virtual Machine or Message Passing Interface.
You can find these testprograms under the website http://www.sentient.de/FMM/ppp.html (additional programm files). The Makefile of additional.zip must be overwritten in directory "test" and the following programm files must be added in directory "test"
.

```
C2.c
1 no_pvm_mpi.c    &    no_pvm_mpi.h
2 line.c          &    line.h
3 lineeps.c       &    lineeps.h
4 grid2D.c        &    grid2D.h
5 star2D.c        &    star2D.h
6 grid3D.c        &    grid3D.h
7 star3D.c        &    star3D.h
```

The C2 testprogramm is executable with a callnumber : each number belongs to a testprogramm.

4 Structure

We have written some testprogramms without using Parallel Virtual Machine or Message Passing Interface. The structure of the particles is different in each testprogramms. We have measured running time with all these testprogramms in VC.Net and in Cygwin too.

4.1 no_pvm_mpi.c

The coordinates of the particles are generated random. (The original dpmta testprogramm with pvm or mpi also generate random particles). In Figure 1. you can see the position of the particles generating by no_pvm_mpi.c. Tabular shows the running time in VisualC++ (VC) and in Cygwin (CW) depending on the number of particles (NumPart).
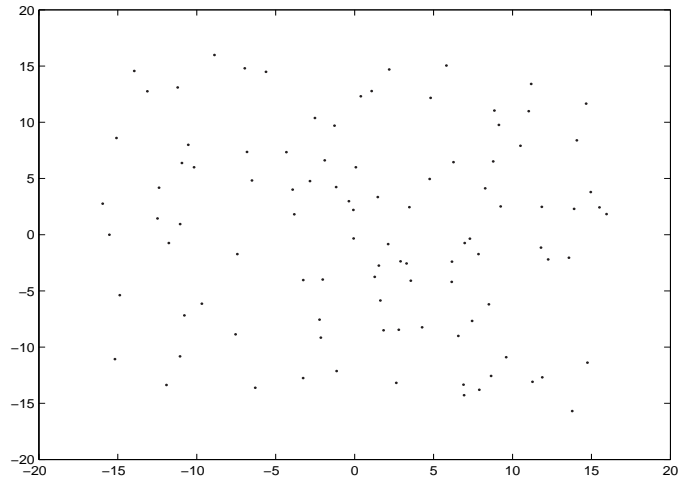
Figure 1: random

NumPart	VC	CW
10	1.593	1.111
25	1.612	1.151
50	1.763	1.211
75	1.752	1.241
100	1.852	1.282
250	2.174	1.552
500	2.624	1.693
750	2.984	1.893
1000	3.325	2.043
2500	4.996	2.864
5000	6.931	3.776
7500	8.443	4.407
10000	9.635	4.797
25000	14.652	6.679
50000	19.588	8.462
75000	22.873	9.694
100000	25.998	10.836
200000	38.271	15.962
300000	60.137	23.404

4.2 line.c

The particles are on a line with equal distance. (Figure 2.) There is a certain space, where the particles can be found. In the first case the distance between two adjacent points is a constant. In the second case the points complete the whole space equally, i.e. the distance is variable. (This holds for lineeps.c, grid2D.c, star2D.c, grid3D.c, star3D.c too.)
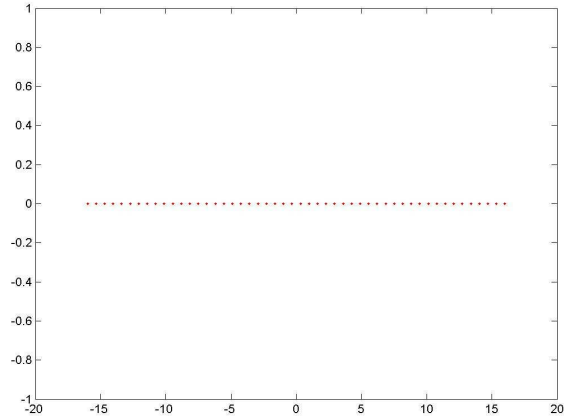

Figure 2: line

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.573	1.563	1.051	1.101
25	1.682	1.543	1.051	1.071
50	1.763	1.562	1.112	1.172
75	1.542	1.612	1.062	1.082
100	1.842	1.543	1.072	1.112
250	1.552	1.542	1.061	1.102
500	1.793	1.542	1.091	1.102
750	1.642	1.572	1.122	1.121
1000	1.733	1.673	1.142	1.141
2500	2.563	1.742	1.392	1.181
5000	5.287	2.173	2.213	1.342
7500	8.152	2.984	3.315	1.592
10000	11.997	3.956	4.056	1.913
25000	28.461	16.304	9.624	6.419
50000	59.006	59.607	18.867	19.929
75000	194.561	132.552	59.276	43.212

4.3 lineeps.c

The particles are on a line with nearly equal distance. The distane between a point in lineeps.c and line.eps can be at most 0.0001. (Figure 3.)
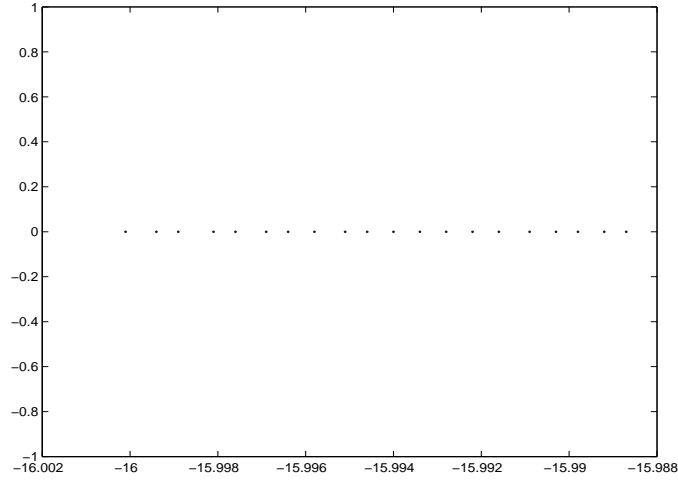

Figure 3: lineeps

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.593	1.643	1.131	1.071
25	1.732	1.622	1.131	1.071
50	1.583	1.662	1.191	1.142
75	1.552	1.893	1.141	1.082
100	1.542	1.623	1.142	1.072
250	1.572	1.823	1.142	1.122
500	1.873	1.763	1.162	1.122
750	1.692	1.663	1.172	1.101
1000	1.853	1.702	1.192	1.131
2500	2.734	1.913	1.442	1.151
5000	6.58	2.494	2.293	1.342
7500	10.335	3.415	3.195	1.622
10000	14.751	4.816	4.095	2.013
25000	39.558	21.381	9.664	5.888
50000	80.656	79.525	19.057	18.396
75000	256.912	176.075	59.546	40.308

4.4 grid2D.c

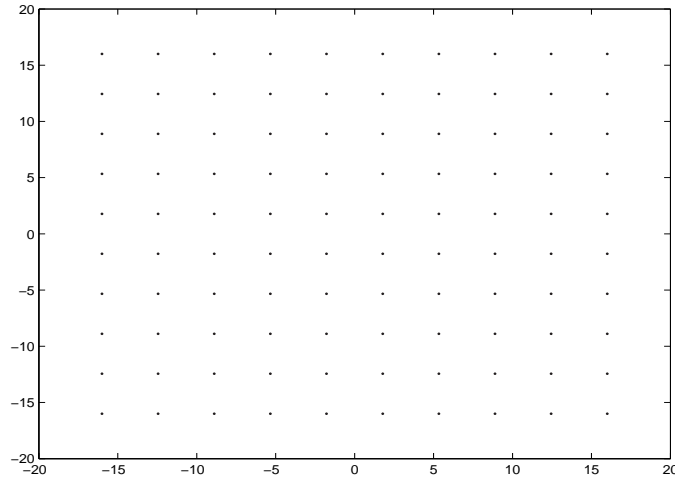The structure of the particles looks like as a grid in the plane (2D). (Figure 4.)

Figure 4: grid2D

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.692	1.703	1.091	1.121
25	1.672	1.692	1.081	1.131
50	1.642	1.742	1.182	1.241
75	1.643	1.792	1.112	1.181
100	1.643	1.823	1.102	1.231
250	1.643	2.063	1.122	1. 342
500	1.683	2.234	1.162	1.412
750	1.742	2.303	1.152	1.402
1000	1.793	2.433	1.182	1.442
2500	2.143	2.464	1.312	1.562
5000	2.864	2.543	1.542	1.712
7500	3.525	2.673	1.802	1.763
10000	4.136	2.825	2.093	1.943
25000	8.442	4.516	3.715	2.924
50000	15.784	10105	6.53	5.037
75000	23.394	19539	9.354	8.072
100000	32.126	31.876	12.718	12.728

4.5 star2D.c

The structure of the particles looks like as a star with random points on the con-
centrical circles. (see Figure 5.) The radius of the innermost circle is constant or
variable. The distance is fixed between two adjacent circles. (this distance equals
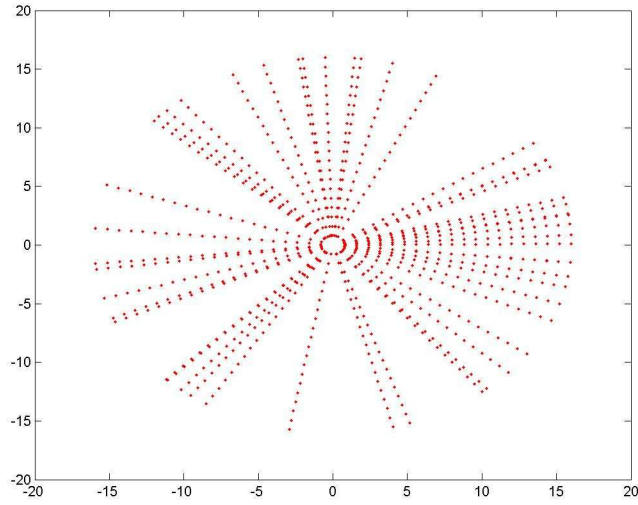the radius of the innermost circle.)

Figure 5: star2D

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.572	1.563	1.061	1.101
25	1.533	1.622	1.051	1.131
50	1.993	1.662	1.112	1.222
75	1.492	1.812	1.062	1.141
100	1.492	1.783	1.072	1.162
250	1.513	2.023	1.061	1.281
500	1.542	2.334	1.081	1.282
750	1.602	2.624	1.091	1.262
1000	1.753	2.484	1.122	1.332
2500	2.804	2.704	1.372	1.402
5000	5.478	3.866	2.053	1.472
7500	8.553	3.596	2.654	1.523
10000	9.524	3.686	3.074	1.683
25000	14.732	8.783	4.366	3.014
50000	21.4	32.758	6.099	7.701
75000	45.686	60.148	12.287	15.162
100000	111	104.831	32.757	25.857

4.6 grid3D.c

The structure of the particles looks like as a grid in the space (3D). The plane section (in the x-y,y-z,x-z planes) looks like as in Figure 4.

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.622	1.593	1.051	1.081
25	2.434	1.663	1.081	1.121
50	1.883	1.892	1.132	1.261
75	1.943	2.043	1.072	1.251
100	1.562	2.203	1.082	1.252
250	1.562	3.104	1.072	1.532
500	1.612	4.646	1.132	1.913
750	1.653	5.167	1.122	2.012
1000	1.742	6.199	1.162	2.183
2500	2.474	11.247	1.412	3.245
5000	5.027	16.073	2.244	4.136
7500	9.344	19.288	3.605	4.837
10000	15.432	21.16	5.528	5.388
25000	76.69	33.188	25.146	7.521
50000			74.849	9.824

4.7   star3D.c

The particles are on concentrical spheres. Figure 6. shows only one sphere with points. The radius R is constant or variable.
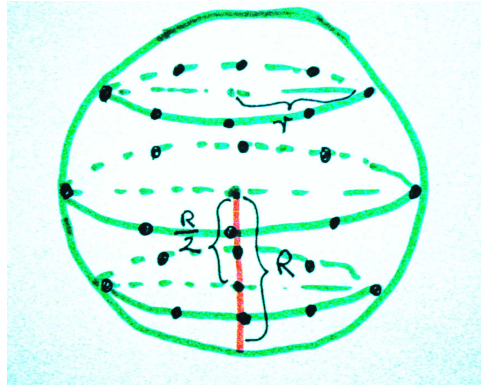

Figure 6:

NumPart	VC-constant	VC-variable	CW-constant	CW-variable
10	1.603	1.613	1.081	1.171
25	1.563	1.603	1.081	1.181
50	1.572	1.663	1.172	1.251
75	1.582	1.733	1.102	1.232
100	1.593	1.712	1.112	1.242
250	1.593	1.973	1.122	1.342
500	1.612	2.253	1.142	1.532
750	1.653	2.464	1.132	1.582
1000	1.733	2.563	1.212	1.643
2500	2.464	2.724	1.442	1.733
5000	5.067	2.985	2.333	1.843
7500	8.983	3.235	3.676	1.933
10000	12.217	3.486	4.767	2.033
25000	19.298	6.619	7.441	3.195
50000	22.732	17.365	8.913	6.949
75000	25.287	35.11	9.924	12.869
100000	29.062	60.508	11.427	21.18
200000	183.226	234.58	63.052	77.281

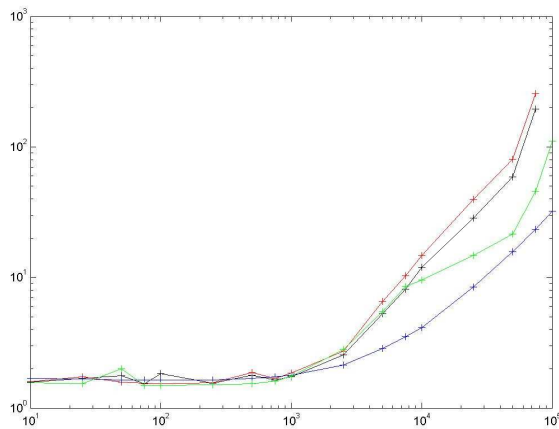Figure 7. shows the running time of some test programms measuring in Visual C++.

Figure 7: running time in VC : line-black, lineeps-red, grid2D-blue, star2D-green

Figure 8. shows the running time of some test programms measuring in Cygwin.

5 Appendix

5.1 Table:content of project files

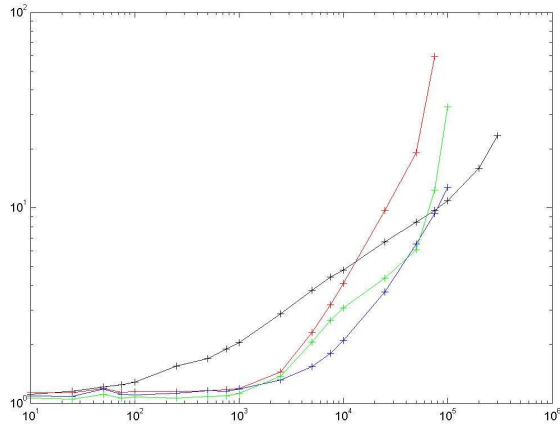The following files must be added to a project file (+).

Figure 8: run time in Cygwin : random-black, lineeps-red, grid2D-blue, star2D-green

Source File		
comm	cpp-Application	c-Application
comm_buffer	+	-
comm_prims	+	-
mpole	cpp-Application	c-Application
ffttest	+	-
mpe_allocC	+	+
mpe_allocLJ	+	+
mpe_fft	+	+
mpe_misc	+	+
mpe_mpoleC	+	+
mpe_mpoleLJ	+	+
mpe_testC	+	-
mpe_testLJ	+	-
src	cpp-Application	c-Application
dpmta_distload	+	-
dpmta_distmisc	+	+
dpmta_distpart	+	-
dpmta_serial	+	+
dpmta_slvcomm	+	+
dpmta_slvcompute	+	+
dpmta_slvglobals	+	+
dpmta_slviter	+	-
dpmta_slvmacro	+	+
dpmta_slvmcalc	+	+
dpmta_slvmkcell	+	+
dpmta_slvmkhl	+	+
dpmta_slvkiil	+	-
dpmta_slvmkil	+	+
dpmta_slvkpcalc	+	+
dpmta_slvscale	+	+
dpmta_timer	+	+

Source File		
test	cpp-Application	c-Application
dpmta_test	+	-
dpmta_testdump	+	-

Header File		
comm	cpp-Application	c-Application
comm_buffer	+	-
comm	+	-
mpole	cpp-Application	c-Application
mpe	+	+
mpe_legendre	+	+
mpe_fftC	+	+
mpe_proto	+	+
mpe_testC	+	+
mpe_testLJ	+	-
mpe_testC	+	-
ffttest	+	-
src	cpp-Application	c-Application
dpmta	+	+
dpmta_cell	+	+
dpmta_config	+	+
dpmta_distload	+	-
dpmta_distmisc	+	+
dpmta_distpart	+	-
dpmta_massage	+	+
dpmta_slvcomm	+	+
dpmta_slvcompute	+	+
dpmta_slvglobals	+	+
dpmta_slviter	+	-
dpmta_slvmacro	+	+
dpmta_slvmcalc	+	-
dpmta_slvmkcell	+	+
dpmta_slvmkhl	+	+
dpmta_slvkiil	+	-
dpmta_slvmkil	+	+
dpmta_slvkpcalc	+	+
dpmta_slvscale	+	+
dpmta_timer	+	+
dpmta_version	+	+
test	cpp-Application	c-Application
dpmta_test	+	-

5.2 Content of additional.zip

```
FastMultipole1.cpp
fftest.h
mpe_testC.h
mpe_testLJ.h
dpmta_test.h
Makefile
C2.c
no_pvm_mpi.c
no_pvm_mpi.h
line.c
line.h
lineeps.c
lineeps.h
grid2D.c
grid2D.h
grid3D.c
grid3D.h
star2D.c
star2D.h
star3D.c
star3D.h
```