fupermod: Functional Performance Models of heterogeneous processors

Version 1.0.0 (Revision 140M)

Generated by Doxygen 1.7.1

Fri Jan 14 2011 10:02:52

Contents

1 Introduction **1**

1.1 Authors . . . 1

2 Installation **1**

3 Tools **2**

3.1 Model builder . . . 2

3.2 Builds models from data points. . . . 3

3.3 Partitioner . . . 3

4 Routines **3**

4.1 Matrix multiplication 1D . . . 3

4.2 Matrix multiplication 2D . . . 5

4.3 Matrix multiplication col . . . 8

5 Module Documentation **15**

5.1 Utilities . . . 15

5.1.1 Variable Documentation . . . 15

5.2 Routines API . . . 15

5.2.1 Detailed Description . . . 15

5.2.2 Define Documentation . . . 16

5.2.3 Typedef Documentation . . . 16

5.3 Functional performance model . . . 17

5.3.1 Enumeration Type Documentation . . . 18

5.3.2 Function Documentation . . . 19

5.4 Functional performance model of heterogeneous multiprocessors and multicores . . . 26

5.5 2D routines API . . . 26

5.5.1 Detailed Description . . . 26

5.5.2 Typedef Documentation . . . 26

5.6 2D functional performance model based on scattered points . . . 29

5.6.1 Function Documentation . . . 29

5.6.2 Variable Documentation . . . 31

6 Class Documentation **31**

6.1 fupermod_2d_data Struct Reference . . . 31

6.1.1 Detailed Description . . . 31

6.1.2 Member Data Documentation . . . 31

6.2 fupermod_2d_distr Struct Reference 32
6.2.1 Detailed Description 32
6.2.2 Member Data Documentation 32
6.3 fupermod_2d_kernel Struct Reference 33
6.3.1 Detailed Description 33
6.3.2 Member Data Documentation 33
6.4 fupermod_2d_model Struct Reference 34
6.4.1 Detailed Description 35
6.4.2 Member Data Documentation 35
6.5 fupermod_2d_routine Struct Reference 35
6.5.1 Detailed Description 36
6.5.2 Member Data Documentation 36
6.6 fupermod_data Struct Reference 37
6.6.1 Detailed Description 37
6.6.2 Member Data Documentation 37
6.7 fupermod_distr Struct Reference 37
6.7.1 Detailed Description 38
6.7.2 Member Data Documentation 38
6.8 fupermod_kernel Struct Reference 38
6.8.1 Detailed Description 38
6.8.2 Member Data Documentation 39
6.9 fupermod_library Struct Reference 39
6.9.1 Detailed Description 39
6.9.2 Member Data Documentation 39
6.10 fupermod_model Struct Reference 41
6.10.1 Detailed Description 41
6.10.2 Member Data Documentation 41
6.11 fupermod_multi_data Class Reference 42
6.11.1 Detailed Description 42
6.12 fupermod_multi_model Class Reference 42
6.12.1 Detailed Description 42
6.13 fupermod_point Struct Reference 43
6.13.1 Detailed Description 43
6.13.2 Member Data Documentation 43
6.14 fupermod_precision Struct Reference 43
6.14.1 Detailed Description 44

6.14.2 Member Data Documentation 44
6.15 fupermod_routine Struct Reference 45
6.15.1 Detailed Description 45
6.15.2 Member Data Documentation 45

1 Introduction

Optimal distribution of computations over the processors of a heterogeneous network can be only achieved based on a performance model of the processors which is able to accurately predict the contribution of computations into the total execution time of any particular parallel or distributed algorithm on the network. The project proposes and investigates realistic performance models of processors in heterogeneous networks, taking into account all the features that may have an impact on the performance, such as architectural and platform heterogeneity of the processors, memory heterogeneity, different levels of integration into the network. The project looks at different aspects of the models including accuracy and adequacy, efficiency of their use, building, maintenance and runtime update.

[1]

1.1 Authors

Alexey Lastovetsky, Vladimir Rychkov, David Clarke

Heterogeneous Computing Laboratory

School of Computer Science and Informatics, University College Dublin

Belfield, Dublin 4, Ireland

http://hcl.ucd.ie

{alexey.lastovetsky, vladimir.rychkov}@ucd.ie, david.clarke.1@ucdconnect.ie

References

[1] A. Lastovetsky and R. Reddy. Data partitioning with a functional performance model of heterogeneous processors. *International Journal of High Performance Computing Applications*, 21:76–90, 2007. 1

2 Installation

```
Information about the PDGEMM application
-----------------------------------------

This application multiplies two dense square
matrices of size n-by-n.

It uses heterogeneous COLUMN_BASED data distribution
of the matrices.

The inputs are (in parameters.h):

1). n     --- size of the matrix
2). (p,q) --- processor grid
```

```
3). kb    --- blocking factor

A restriction on n:

n must be a lcm(p*kb,q*kb).

For example, (p,q)=(3,2) and kb=5, then n must be a multiple
of 30.

Each process writes its output to a file of the form
<hostname>_I_J.txt where (I,J) are the coordinates of the
process.
There will be two lines:
<--COMM--> represents communication times
<**COMP**> represents computation times

The distribution strategy used for computations is
HDS:

In this strategy, one process is executed per
computer (the computer may have one or more processors).
The volume of data allocated to a computer is proportional
to the speed of the computer.

HOW TO RUN
----------

shell$ mpicc -I<hmpi install directory>/include -o mxm mxm.c -L<hmpi install dire
      ctory>/lib -lhmpi -lmpc

You must link the BLAS libraries.

Use 'mpirun' to run the program.
```

3 Tools

- Model builder
- Builds models from data points.
- Partitioner

3.1 Model builder

Builds full functional performance model. Outline:

```
open output file, filename from routine name and host name
Print header info from options & subopts
Loop over different problem sizes, using different methods.
   Initialise routine
   loop over reptitions
       start time
           kernel
       end time
   Finialise routine
   Do something with the array of times, mean, max, min, student-t...
   Print out times to file
End Loop
close file
```

3.2 Builds models from data points.

Supports different types of interpolation (recommended for the models with few data points) and least squares B-spline approximation (recommended for the models with large number of data points)

3.3 Partitioner

Partitions 1D workload using 1 of 6 algorithms. Then routine is executed. Algorithms:

1. Partial functional performance model
2. Full functional performance model
3. Constant performance model 1 (small kernel)
4. Constant performance model 2 (homogeneous kernel)
5. Homogeneous distribution
6. Manual distribution

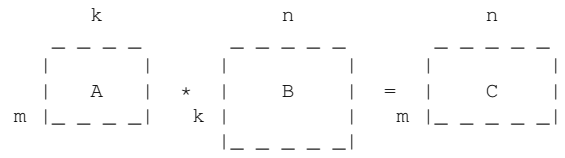
4 Routines

Routines implement the interface described in Routines API or 2D routines API

- Matrix multiplication 1D
- Matrix multiplication 2D
- Matrix multiplication col

4.1 Matrix multiplication 1D

Matrices A and C to distribute. Size m to partition



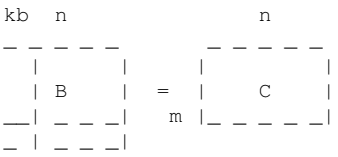

```
#include "config.h"
#include "fupermod_routine.h"
#include <stdlib.h>
#include <stdio.h>
#include <cblas.h>
#include "fupermod.h"

/* Library */

int n = 0;
int k = 0;

int mxm_1d_conf(FILE* stream) {
    // todo:
    fprintf(stream, "#Suboptions: n %d, k %d\n", n, k);
```

```
    return 0;
}

int mxm_1d_load(MPI_Comm comm, char* subopts) {
    char* tokens[] = {"n", "k"};
    char* value;
    while (*subopts != '\0') {
        switch (getsubopt(&subopts, tokens, &value)) {
        case 0:
            n = atoi(value);
            break;
        case 1:
            k = atoi(value);
            break;
        default:
            fprintf(stderr, "%s: unknown suboption %s\n", __func__, value);
            break;
        }
    }
    return FUPERMOD_SUCCESS;
}

int mxm_1d_unload(MPI_Comm comm) {
    return FUPERMOD_SUCCESS;
}

double mxm_1d_complexity(int d) {
    return 2.0 * d * n * k;
}

/* Routine/Benchmark */

int m = 0;
double* A = NULL;
double* B = NULL;
double* C = NULL;

int mxm_1d_initialize(MPI_Comm comm, int d) {
    m = d;
    A = (double*) malloc(sizeof(double) * m * k);
    B = (double*) malloc(sizeof(double) * k * n);
    C = (double*) malloc(sizeof(double) * m * n);
    if (A == NULL || B == NULL || C == NULL) {
        fprintf(stderr, " *** Cannot allocate A, B or C, m=%d, n=%d, k=%d\n", m,
      n, k);
        return FUPERMOD_FAIL;
    }
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < k; j++)
            A[i * k + j] = i * k + j;
    for (i = 0; i < k; i++)
        for (j = 0; j < n; j++)
            B[i * n + j] = i * n + j;
    return FUPERMOD_SUCCESS;
}

int mxm_1d_execute(MPI_Comm comm) {
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, k, 1, A, k, B, n
      , 0, C, n);
    return FUPERMOD_SUCCESS;
}

int mxm_1d_finalize(MPI_Comm comm) {
    m = 0;
    free(A); A = NULL;
    free(B); B = NULL;
```

```
    free(C); C = NULL;
    return FUPERMOD_SUCCESS;
}

fupermod_routine fupermod_routine_instance = {
    {
        1,
        "mxm1d",
        mxm_1d_conf,
        mxm_1d_load,
        mxm_1d_unload
    },
    mxm_1d_complexity,
    mxm_1d_initialize,
    mxm_1d_execute,
    mxm_1d_finalize,
    {
        mxm_1d_complexity,
        mxm_1d_initialize,
        mxm_1d_execute,
        mxm_1d_finalize
    }
};
```

4.2 Matrix multiplication 2D

Matrices A, B, C to distribute $p * q$ (number of processors must be equal to $p * q$!). Sizes m, n to partition. All sizes in number of blocks (not number of rows or columns), kb is a block size




```
#include "config.h"
#include "fupermod_2d_routine.h"
#include <stdlib.h>
#include <stdio.h>
#include <cblas.h>
#include <memory.h>
#include <gsl/gsl_sort_vector.h>

/* Library */

int p = 0;
int q = 0;
int k = 0;
int kb = 0;

int mxm_2d_conf(FILE* stream) {
    // todo:
    fprintf(stream, "#Suboptions: p %d, q %d, k %d, kb %d\n", p, q, k, kb);
    return 0;
}

int mxm_2d_load(MPI_Comm comm, char* subopts) {
    char* tokens[] = {"p", "q", "k", "kb"};
    char* value;
    while (*subopts != '\0') {
        switch (getsubopt(&subopts, tokens, &value)) {
        case 0:
            p = atoi(value);
            break;
```

```
        case 1:
            q = atoi(value);
            break;
        case 2:
            k = atoi(value);
            break;
        case 3:
            kb = atoi(value);
            break;
        default:
            fprintf(stderr, "%s: unknown suboption %s\n", __func__, value);
            break;
        }
    }
    int size = 0;
    if (comm != MPI_COMM_WORLD)
        MPI_Comm_size(comm, &size);
    if (size != p * q) {
        fprintf(stderr, "%s: communicator size != p * q\n", __func__);
        return -1;
    }
    return 0;
}

int mxm_2d_unload(MPI_Comm comm) {
    return 0;
}

/* Routine */

int m = 0;
int n = 0;
double* A = NULL;
double* B = NULL;
double* C = NULL;

double mxm_2d_complexity(int m, int n) {
    return 2.0 * m * kb * n * kb * k * kb;
}

int mxm_2d_initialize(MPI_Comm comm, int m_,  int n_) {
    m = m_;
    n = n_;
    A = (double*) malloc(sizeof(double) * m * kb * k * kb);
    B = (double*) malloc(sizeof(double) * k * kb * n * kb);
    C = (double*) malloc(sizeof(double) * m * kb * n * kb);
    if (A == NULL || B == NULL || C == NULL) {
        fprintf(stderr, " *** Cannot allocate A, B or C, m=%d, n=%d, k=%d\n", m *
       kb, n * kb, k * kb);
        return -1;
    }
    int i, j;
    for (i = 0; i < m * kb; i++)
        for (j = 0; j < k * kb; j++)
            A[i * k * kb + j] = i * k * kb + j;
    for (i = 0; i < k * kb; i++)
        for (j = 0; j < n * kb; j++)
            B[i * n * kb + j] = i * n * kb + j;
    return 0;
}

int mxm_2d_execute(MPI_Comm comm) {
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m * kb, n * kb, k * kb
      , 1, A, k * kb, B, n * kb, 0, C, n * kb);
    return 0;
}
```

```
int mxm_2d_finalize(MPI_Comm comm) {
    m = n = 0;
    free(A); A = NULL;
    free(B); B = NULL;
    free(C); C = NULL;
    return 0;
}

/* kernel */

double mxm_2d_kernel_complexity(int m, int n) {
    return 2.0 * m * kb * n * kb * kb;
}

int mxm_2d_kernel_complexity_derivatives(int m, int n, double* dm, double* dn) {
    *dm = 2.0 * kb * n * kb * kb;
    *dn = 2.0 * m * kb * kb * kb;
    return 0;
}

int mxm_2d_x0(int M, int N, int size, double* m, double* n) {
    if (size != p * q)
        return -1;
    int i, j;
    for (i = 0; i < p; i++)
        for (j = 0; j < q; j++) {
            m[i * q + j] = M / p;
            n[i * q + j] = N / q;
        }
    return 0;
}

int mxm_2d_f(int size, double* m, double* n, int M, int N, double* f) {
    if (size != p * q)
        return -1;
    int i, j;
    // m(0, 0) + m(1, 0) + ... + m(p - 1, 0) = M
    f[0] = M;
    for (i = 0; i < p; i++) {
        f[0] -= m[i * q];
        // n(i, 0) + n(i, 1) + ... + n(i, q - 1) = N
        f[i + 1] = N;
        for (j = 0; j < q; j++)
            f[i + 1] -= n[i * q + j];
        // m(i, 0) = m(i, 1) = ... = m(i, q - 1)
        for (j = 1; j < q; j++)
            f[1 + p + i * (q - 1) + j - 1] = m[i * q] - m[i * q + j];
    }
    return 0;
}

int mxm_2d_df(int size, double* m, double* n, int M, int N, double* J) {
    if (size != p * q)
        return -1;
    // zeros
    memset(J, 0, sizeof(double) * (size + 1) * (2 * size));
    int i, j;
    // f(0)
    for (i = 0; i < p; i++) {
        J[i * q] = -1;
        // f(1) ... f(1 + p)
        for (j = 0; j < q; j++)
            J[2 * size * (i + 1) + size + i * q + j] = -1;
        // the rest
        for (j = 1; j < q; j++) {
            J[2 * size * (1 + p + i * (q - 1) + j - 1) + i * q] = 1;
            J[2 * size * (1 + p + i * (q - 1) + j - 1) + i * q + j] = -1;
```

```
        }
    }
    return 0;
}

int mxm_2d_remainders(int M, int N, int size, double* m, double* n, int* new_m, i
      nt* new_n) {
    //todo: implement this to complete 2D models based on scattered points
/*
    int D_sum = 0;
    int i;
    for (i = 0; i < size; i++)
        D_sum += gsl_vector_get(solver->x, i);
    // sort in asc order
    gsl_permutation* p = gsl_permutation_alloc(size);
    gsl_sort_vector_index(p, solver->x);
    gsl_multiroot_fdfsolver_free(solver);
    // remainder D - D_sum < size
    i = size - 1;
    while (D_sum < D) {
        d[p->data[i--]]++;
        D_sum++;
    }
    gsl_permutation_free(p);
*/
    return 0;
}

int mxm_2d_kernel_execute(MPI_Comm comm) {
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m * kb, n * kb, kb, 1,
       A, kb, B, n * kb, 0, C, n * kb);
    return 0;
}

fupermod_2d_routine fupermod_routine_instance = {
    {
        2,
        "mxm_2d",
        mxm_2d_conf,
        mxm_2d_load,
        mxm_2d_unload
    },
    mxm_2d_complexity,
    mxm_2d_initialize,
    mxm_2d_execute,
    mxm_2d_finalize,
    {
        mxm_2d_kernel_complexity,
        mxm_2d_kernel_complexity_derivatives,
        mxm_2d_x0,
        mxm_2d_f,
        mxm_2d_df,
        mxm_2d_remainders,
        mxm_2d_initialize,
        mxm_2d_kernel_execute,
        mxm_2d_finalize
    }
};
```

4.3 Matrix multiplication col

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <cblas.h>
```

```
#include <mpi.h>
#include "../../src/fupermod.h"
#include "../../src/mm_col_dist.h"
#include "mxm_col.h"

#define VERBOSE_MATRIX 0
#define TEST_RESULT 0
#define dx(i) idist->d[i].x
#define dy(i) idist->d[i].y
#define dw(i) idist->d[i].w
#define dh(i) idist->d[i].h
#define di(k) idist->d[k].i
#define dj(k) idist->d[k].j

double MXM_CONSTANT_NUMBER1 = 1.5;
double MXM_CONSTANT_NUMBER2 = 2.5;
int MPI_MSG_TAG_FOR_A = 0xf1;
int MPI_MSG_TAG_FOR_B = 0xf2;

int mxm_col(MPI_Comm comm, mm_col_i_distribution idist, mxm_times* times){
    int me;
    MPI_Comm_rank(comm, &me);

    int myx, myy, myw, myh, myi, myj;
    myx = idist->d[me].x;
    myy = idist->d[me].y;
    myw = idist->d[me].w;
    myh = idist->d[me].h;
    myi = idist->d[me].i;
    myj = idist->d[me].j;

    int kb = idist->kb;
    int nproc = idist->nproc;
    int N = idist->N;
    int rc;

    MPI_Comm col_comm;
    rc = MPI_Comm_split(comm, myj, myi, &col_comm);
    if (rc != MPI_SUCCESS) {
        printf("Error splitting groups\n");
    }


    double* comm_time_a = (double*)malloc(sizeof(double)*N);
    double* comm_time_b = (double*)malloc(sizeof(double)*N);
    double* comp_time = (double*)malloc(sizeof(double)*N);
    double* extra_time = (double*)malloc(sizeof(double)*N);

    times->comp_time = comp_time;
    times->comm_time_a = comm_time_a;
    times->comm_time_b = comm_time_b;
    times->extra_time = extra_time;
    struct timeval startt, endt, start_tot, end_tot;

    // For non blocking MPI_Irecv
    MPI_Status *statusesS, *statusesR;
    MPI_Request *requestsS, *requestsR;
    requestsS = (MPI_Request*)malloc(sizeof(MPI_Request)*(nproc-1));
    requestsR = (MPI_Request*)malloc(sizeof(MPI_Request)*(nproc-1));
    statusesS=(MPI_Status*)malloc(sizeof(MPI_Status)*(nproc-1));
    statusesR=(MPI_Status*)malloc(sizeof(MPI_Status)*(nproc-1));
    //non-blocking index
    int recv_count, send_count = 0;

    // initialise A, B, C, WA, WB
    double *A, *B, *C, *WA, *WB;
    A = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
```

```
B = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
C = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
// working space:
WA = (double*) malloc(sizeof(double) * (myh * kb * kb));
WB = (double*) malloc(sizeof(double) * (myw * kb * kb));
// Did it work?
if (A == NULL || B == NULL || C == NULL || WA == NULL || WB == NULL) {
    fprintf(stderr, "P%d Failed to allocate matrix\n"
            "P%d myw = %d, myh = %d, kb = %d\n"
            "p%d Exiting\n",me, me, myw, myh, kb, me);
    free(A);
    free(B);
    free(C);
    free(WA);
    free(WB);
    return FUPERMOD_ALLOC_FAIL;
}
// Fill A, B. Zero C
int i, j;
if (TEST_RESULT) {
    for (i = 0; i < myh * kb; i++) {
        for (j = 0; j < myw * kb ; j++) {
            A[i * myw * kb + j] = N * kb * (myy * kb + i) + myx * kb + j;
            //      printf("%f", A[i])
            B[i * myw * kb + j] = N * kb * (myy * kb + i) + myx * kb + j;
            C[i * myw * kb + j] = 0.0;
        }
    }
} else {
    for (i = 0; i < myh * kb * myw * kb; i++) {
        A[i] = MXM_CONSTANT_NUMBER1;
        B[i] = MXM_CONSTANT_NUMBER2;
        C[i] = 0.0;
    }
}

// fprintf(stderr, "P%d Starting k loop\n",me);
// Main k loop - moving pivrow & pivcol
gettimeofday(&start_tot, NULL);
int k;
for (k = 0; k < N; k++) {
    double* A_pt = WA;
    double* B_pt = WB;
    int lda = kb, ldb = myw * kb;
    comm_time_a[k] = 0;
    comm_time_b[k] = 0;
    recv_count = 0;
    send_count = 0;

    // Find j index of procs holding pivcol
    int pivcol_j = -1;
    for (i = 0; i < nproc; i++) {
        if (k >= dx(i) && k < dx(i)+dw(i)) {
            pivcol_j = dj(i);
            break;
        }
    }
    // Am i pivcol?
    int pivcol = 0;
    if (myj == pivcol_j) {
        pivcol = 1;
        A_pt = A + (k - myx) * kb; // point to local A.
        lda = myw * kb; // lda changes.
    }
    int pivrow_proc = -1;
    for (i = 0; i < nproc; i++) {
        if ((dj(i) == myj) && k >= dy(i) && k < dy(i)+dh(i)) {
```

```
          pivrow_proc = i;
          break;
      }
  }
  if (pivcol_j < 0) {
      fprintf(stderr, "Error, Problem with pivcol\n");
  }
  if (pivrow_proc < 0) {
      fprintf(stderr, "P%d: Error, Problem with pivrow_proc\n", me);
  //  return FUPERMOD_MPI_FAIL;
  }
  //MPI_Barrier(comm);

  // Send/recv    pivcol of A
  // test intersection with each proc.
  for (i = 0; i < nproc; i++) {
      //MPI_Barrier(comm);
      if ((pivcol || dj(i) == pivcol_j) && i != me && // either i'm pivcol
or proc i is
              (myy < dy(i) + dh(i)) && (myy + myh > dy(i))) { // there is a
n overlap

          //start send/recv from
          int start = dy(i) - myy;
          if (start < 0) {
              start = 0;
          }
          int length;
          if (dy(i) + dh(i) > myy + myh) {
              length = myh - start;
          } else {
              length = dy(i) + dh(i) - myy - start;
          }
          gettimeofday(&startt, NULL);
          if (pivcol) { // send
              MPI_Datatype    fupermod_col;
              MPI_Type_vector(length * kb, kb, myw * kb, MPI_DOUBLE, &fuper
mod_col);
              MPI_Type_commit(&fupermod_col);

              rc = MPI_Isend(&A[(start * kb) * (myw *kb) + (k-myx) * kb], 1
, fupermod_col, i, MPI_MSG_TAG_FOR_A, comm, requestsS+send_count);
              send_count++;
              if (rc != MPI_SUCCESS) {
                  printf("Error Sending A\n");
                  return FUPERMOD_MPI_FAIL;
              }
              MPI_Type_free(&fupermod_col);
          } else { // recv
              rc = MPI_Irecv((WA+start * kb * kb), length * kb * kb, MPI_DO
UBLE, i, MPI_MSG_TAG_FOR_A, comm, requestsR+recv_count);
              recv_count++;
              if (rc != MPI_SUCCESS) {
                  printf("Error Recving A\n");
                  return FUPERMOD_MPI_FAIL;
              }
          }
          gettimeofday(&endt, NULL);
          comm_time_a[k] += endt.tv_sec - startt.tv_sec + (endt.tv_usec - s
tartt.tv_usec)/1.0E6;
      }
  }
  // send/recv pivrow of B
  gettimeofday(&startt, NULL);
  if (pivrow_proc == me) {
      rc = MPI_Bcast((B+(k-myy)*kb*kb*myw), myw * kb * kb, MPI_DOUBLE, myi,
 col_comm);
```

```
        B_pt = B + ((k-myy) * kb) * kb * myw; // Set for local dgemm
    } else {
        rc = MPI_Bcast(WB, myw * kb * kb, MPI_DOUBLE, di(pivrow_proc), col_co
mm);
    }
    if (rc != MPI_SUCCESS) {
        printf("Error Bcat'n B\n");
        return FUPERMOD_MPI_FAIL;
    }
    gettimeofday(&endt, NULL);
    comm_time_b[k] += endt.tv_sec - startt.tv_sec + (endt.tv_usec - startt.tv
_usec)/1.0E6;

    // Wait for all recv's to complete.
    gettimeofday(&startt, NULL);
    MPI_Waitall(recv_count, requestsR, statusesR);
    //      MPI_Waitall(send_count, requestsS, statusesS);
    gettimeofday(&endt, NULL);
    comm_time_a[k] += endt.tv_sec - startt.tv_sec + (endt.tv_usec - startt.tv
_usec)/1.0E6;

    if (VERBOSE_MATRIX) {
        if (me == 1) {
            printf("P%d A(%d,%d):\n", me, myh * kb, kb);
            for (i = 0; i < myh * kb; i++) {
                for (j = 0; j < kb; j++) {
                    printf("  %3.3f", A_pt[i * lda + j]);
                }
                printf("\n");
            }
        }
        if (me == 1) {
            printf("P%d B(%d,%d):\n", me, myh * kb, kb);
            for (i = 0; i < kb; i++) {
                for (j = 0; j < ldb; j++) {
                    printf("  %3.3f", B_pt[i * ldb + j]);
                }
                printf("\n");
            }
        }
    }
    gettimeofday(&startt, NULL);
    //MPI_Barrier(comm);
    gettimeofday(&endt, NULL);
    extra_time[k] = endt.tv_sec - startt.tv_sec + (endt.tv_usec - startt.tv_u
sec)/1.0E6;
    // multiply WA x WB = C
    gettimeofday(&startt, NULL);
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, myh * kb, myw * kb
, kb, 1, A_pt,  lda, B_pt, ldb, 1, C, myw * kb);
    //MPI_Barrier(comm);
    gettimeofday(&endt, NULL);
    comp_time[k] = endt.tv_sec - startt.tv_sec + (endt.tv_usec - startt.tv_us
ec)/1.0E6;
    if (VERBOSE_MATRIX) {
        if (me == 0) {
            printf("P%d C(%d,%d):\n", me, myh * kb, myw * kb);
            for (i = 0; i < myw * kb; i++) {
                for (j = 0; j < myh * kb; j++) {
                    printf("  %3.3f", C[i * myw * kb + j]);
                }
                printf("\n");
            }
        }
    }
}// end k loop
gettimeofday(&end_tot, NULL);
```

```
    times->tot_time = end_tot.tv_sec - start_tot.tv_sec + (end_tot.tv_usec - star
      t_tot.tv_usec)/1.0E6;

    if (TEST_RESULT) {
        double *At, *Bt, *Ct;
        int Nb = N * kb;
        At = (double*) malloc(sizeof(double) * (Nb * Nb));
        Bt = (double*) malloc(sizeof(double) * (Nb * Nb));
        Ct = (double*) malloc(sizeof(double) * (Nb * Nb));

        for (i = 0; i < (Nb * Nb); i++) {
            At[i] = i;
            Bt[i] = i;
            Ct[i] = 0.0;
        }
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, Nb, Nb, Nb, 1, At,
          Nb, Bt, Nb, 1, Ct, Nb);
        int err = 0;
        for (i = 0; i < myh * kb; i++) {
            for (j = 0; j < myw * kb; j++) {
                if (((C[i * myw * kb + j] - Ct[(i+myy * kb) * Nb + (j + myx * kb)
      ]) / C[i * myw * kb + j]) > 0.01)
                    err++;
            }
        }
        fprintf(stderr, "P%d: Test MXM, error: %d\n", me, err);
        free(At);
        free(Bt);
        free(Ct);
    }
    //finish all communication before exiting
    MPI_Waitall(send_count, requestsS, statusesS);
    free(requestsS);
    free(statusesS);
    free(requestsR);
    free(statusesR);
    free(A); A = NULL;
    free(B);
    free(C);
    free(WA);
    free(WB);
    return FUPERMOD_SUCCESS;
}

int mxm_col_benchmark(MPI_Comm comm, int area, int kb, double* time, double* comp
      lexity){
    int me;
    MPI_Comm_rank(comm, &me);
    int myw, myh;
// add 10% for extra overhead
    myw = myh = sqrt(1.1 * area);
    int length = myh;
    MPI_Status statusesR;
    MPI_Request requestsS, requestsR;
    int i;
    int k_rand = 0;
    int iters = 1;
    *complexity = 2.0 * myw * myh *  kb * kb * kb;
    // initialise A, B, C, WA, WB
    double *A, *B, *C, *WA, *WB;
    A = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
    B = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
    C = (double*) malloc(sizeof(double) * (myw * kb * myh * kb));
    // working space:
    WA = (double*) malloc(sizeof(double) * (myh * kb * kb));
    WB = (double*) malloc(sizeof(double) * (myw * kb * kb));
    // Did it work?
```

```
if (A == NULL || B == NULL || C == NULL || WA == NULL || WB == NULL) {
    fprintf(stderr, "P%d Failed to allocate matrix\n"
            "P%d myw = %d, myh = %d, kb = %d\n"
            "p%d Exiting\n",me, me, myw, myh, kb, me);
    *time = 1E9;
    free(A);
    free(B);
    free(C);
    free(WA);
    free(WB);
    return FUPERMOD_ALLOC_FAIL;
}
// Fill A, B. Zero C
for (i = 0; i < myh * kb * myw * kb; i++) {
    A[i] = MXM_CONSTANT_NUMBER1;
    B[i] = MXM_CONSTANT_NUMBER2;
    C[i] = 0.0;
}
// Pull WA, WB into main memory, possibly push out A,B,C
for (i = 0; i < myh * kb * kb; i++) {
    WA[i] = 0.0;
}
for (i = 0; i < myw * kb * kb; i++) {
    WB[i] = 0.0;
}

struct timeval startt, endt;
// Start timeing
gettimeofday(&startt, NULL);
int k;
for (k = 0; k < iters; k++) {
    k_rand = k % myw;
    MPI_Datatype    fupermod_col;
    MPI_Type_vector(length * kb, kb, myw * kb, MPI_DOUBLE, &fupermod_col);
    MPI_Type_commit(&fupermod_col);

    int rc;
    rc = MPI_Isend(&A[(k_rand) * kb], 1, fupermod_col, me, MPI_MSG_TAG_FOR_A,
   comm, &requestsS);
    if (rc != MPI_SUCCESS) {
        printf("Error Sending A\n");
        return FUPERMOD_MPI_FAIL;
    }
    MPI_Type_free(&fupermod_col);
    rc = MPI_Irecv(WA, length * kb * kb, MPI_DOUBLE, me, MPI_MSG_TAG_FOR_A, c
  omm, &requestsR);
    if (rc != MPI_SUCCESS) {
        printf("Error Recving A\n");
        return FUPERMOD_MPI_FAIL;
    }

    double* B_local = &B[(k_rand)*kb*kb*myw];
    for (i = 0; i < myw * kb * kb; i++) {
        WB[i] = B_local[i];
    }
    MPI_Waitall(1, &requestsR, &statusesR);

    // multiply WA x WB = C
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, myh * kb, myw * kb
  , kb, 1, WA,    kb, WB, myw * kb, 1, C, myw * kb);
    //MPI_Barrier(comm);
}
gettimeofday(&endt, NULL);
*time = (endt.tv_sec - startt.tv_sec + (endt.tv_usec - startt.tv_usec)/1.0E6)
   / iters;
free(A);
free(B);
```

```
    free(C);
    free(WA);
    free(WB);
    return FUPERMOD_SUCCESS;
}
```

5 Module Documentation

5.1 Utilities

Variables

- int fupermod_verbose

5.1.1 Variable Documentation

5.1.1.1 int fupermod_verbose

Verbose level (default: 0 - none)

5.2 Routines API

Classes

- struct fupermod_library
- struct fupermod_kernel
- struct fupermod_routine

Defines

- #define FUPERMOD_SUCCESS 0
- #define FUPERMOD_ROUTINE_INSTANCE "fupermod_routine_instance"

Typedefs

- typedef double(∗ fupermod_complexity)(int d)
- typedef int(∗ fupermod_initialize)(MPI_Comm comm, int d)
- typedef int(∗ fupermod_execute)(MPI_Comm comm)
- typedef int(∗ fupermod_finalize)(MPI_Comm comm)

5.2.1 Detailed Description

Each routine must be implemented in a shared library as an instance of the Routines API structure. The fupermod tools will search for a symbol of this instance.

5.2.2 Define Documentation

5.2.2.1 #define FUPERMOD_SUCCESS 0

Standard return values from functions

5.2.2.2 #define FUPERMOD_ROUTINE_INSTANCE "fupermod_routine_instance"

Symbol to search for in a routine shared library

5.2.3 Typedef Documentation

5.2.3.1 typedef double(∗ fupermod_complexity)(int d)

Routine/kernel complexity. Must linearly depend on the problem size

Parameters

d problem size

Returns

complexity

5.2.3.2 typedef int(∗ fupermod_initialize)(MPI_Comm comm, int d)

Initializes routine/kernel. Allocates data for benchmarking, which are stored in global variables.

Parameters

comm MPI communicator

d problem size

Returns

return code

5.2.3.3 typedef int(∗ fupermod_execute)(MPI_Comm comm)

Executes routine/kernel

Parameters

comm MPI communicator

Returns

return code

5.2.3.4 typedef int(∗ fupermod_finalize)(MPI_Comm comm)

Finalizes routine/kernel

Parameters

comm MPI communicator

Returns

return code

5.3 Functional performance model

Classes

- struct fupermod_point
- struct fupermod_data
- struct fupermod_precision
- struct fupermod_model
- struct fupermod_distr

Enumerations

- enum fupermod_benchmark_mode { sc_mp, mc_mc, mc_mp }
- enum fupermod_interp_type { fupermod_interp_linear, fupermod_interp_akima }
- enum fupermod_algorithm {
  partial, full, constant1, constant2,
  homogeneous, manual }
- enum fupermod_section

Functions

- int fupermod_point_read (FILE ∗stream, fupermod_point ∗point)
- void fupermod_point_write (FILE ∗stream, fupermod_point ∗point, fupermod_library ∗routine)
- fupermod_data ∗ fupermod_data_alloc ()
- void fupermod_data_free (fupermod_data ∗data)
- fupermod_data ∗ fupermod_data_read (FILE ∗stream)
- void fupermod_data_write (FILE ∗stream, fupermod_data ∗data, fupermod_library ∗routine)
- void fupermod_data_write_head (FILE ∗stream, fupermod_library ∗routine)
- int fupermod_data_index (fupermod_data ∗data, int d)
- double fupermod_data_time (fupermod_data ∗data, int d)
- int fupermod_data_add (fupermod_data ∗data, fupermod_point point)
- double fupermod_ci (double cl, int reps, double ∗t)
- fupermod_data ∗ fupermod_do_benchmark (fupermod_kernel ∗kernel, MPI_Comm comm, int d, int sync)
- double ∗ fupermod_do_benchmark_repitition (fupermod_kernel ∗kernel, MPI_Comm comm, int d, int reps, int sync)
- int fupermod_comm_split (MPI_Comm comm, MPI_Comm ∗newcomm)

- fupermod_point fupermod_do_benchmark_ (fupermod_kernel *kernel, MPI_Comm comm_global, MPI_Comm comm_node, int rank, int d, fupermod_precision precision, fupermod_benchmark_mode mode)
- FILE * fupermod_data_fopen (const char *file_path, fupermod_library *routine, const char mode, int rank, fupermod_benchmark_mode benchmark_mode)
- FILE * fupermod_data_file_open_host (const char *file_path, const char *hostname, fupermod_library *routine, const char mode)
- void trim_hostname (char *hostname)
- int read_machinefile (char *machinefile, char ***hostnames, int *size)
- double fupermod_model_time (fupermod_model *model, const int d)
- void fupermod_print_models (FILE *stream, const int size, fupermod_model **models, const int D)
- fupermod_model * fupermod_model_interp_alloc (fupermod_data *data, int D, fupermod_kernel *kernel, fupermod_interp_type type, int smooth)
- void fupermod_model_interp_free (fupermod_model *model)
- void fupermod_model_multifit_free (fupermod_model *model)
- fupermod_distr * fupermod_distr_alloc (int size, int D)
- void fupermod_distr_free (fupermod_distr *distr)
- int fupermod_partition (int D, int size, fupermod_model **models, int iters_hybrj, double eps_hybrj, fupermod_distr *distr)
- int fupermod_partition_geom (int D, int size, fupermod_model **models, fupermod_distr *distr)
- int fupermod_partition_cpm (int D, int size, double *times, fupermod_distr *distr)
- int fupermod_partition_dynamic (MPI_Comm comm, int root, int iters, int eps, fupermod_kernel *kernel, int iters_hybrj, double eps_hybrj, fupermod_distr *distr, fupermod_model **models, char **emulate_hosts, fupermod_library *emulate_routine)
- int fupermod_balance (MPI_Comm comm, int root, fupermod_section section, struct timeval *start, int D, double threshold, fupermod_distr *distr, fupermod_model **models)
- void fupermod_balance_init (MPI_Comm comm, int root, fupermod_algorithm algorithm, int D, char *info)
- void fupermod_balance_finalise (int size, fupermod_model **models)

5.3.1 Enumeration Type Documentation

5.3.1.1 enum fupermod_benchmark_mode

Select which model generating mode to use

Enumerator:

sc_mp Single core host, one model per host

mc_mc Multi core host, one model per core

mc_mp Multi core host, one model per host

5.3.1.2 enum fupermod_interp_type

Types of interpolation used in models

Enumerator:

fupermod_interp_linear Linear interpolation

fupermod_interp_akima Akima interpolation

5.3.1.3 enum fupermod_algorithm

Select which distribution algorithm to use

Enumerator:

partial Dynamicaly built fpm

full Pre-prepared full fpm

constant1 Small benchmark

constant2 Full size benchmark / homogeneous initial partitioning

homogeneous No balancing

manual User set

5.3.1.4 enum fupermod_section

Marks a beginning or an end of the code section to balance

5.3.2 Function Documentation

5.3.2.1 int fupermod_point_read (FILE * *stream,* fupermod_point * *point*)

Reads data point from file

5.3.2.2 void fupermod_point_write (FILE * *stream,* fupermod_point * *point,* fupermod_library * *routine*)

Writes data point to file

5.3.2.3 fupermod_data* fupermod_data_alloc ( )

Allocates the data

5.3.2.4 void fupermod_data_free (fupermod_data * *data*)

Frees the data

5.3.2.5 fupermod_data* fupermod_data_read (FILE * *stream*)

Reads data from a file

5.3.2.6 void fupermod_data_write (FILE * *stream,* fupermod_data * *data,* fupermod_library * *routine*)

Writes data to a file

5.3.2.7 void fupermod_data_write_head (FILE * *stream,* fupermod_library * *routine*)

Writes header info to a file

5.3.2.8 int fupermod_data_index (fupermod_data * *data,* int *d*)

Returns an index of the data point or -1

5.3.2.9 double fupermod_data_time (fupermod_data * *data,* int *d*)

Looksup data for a particular point and returns time if found or 0

5.3.2.10 int fupermod_data_add (fupermod_data * *data,* fupermod_point *point*)

Adds a data point

Parameters

data data

point point

Returns

index or -1

5.3.2.11 double fupermod_ci (double *cl,* int *reps,* double * *t*)

Returns a confidence interval that contains the average execution time with a certain probability: $Pr(|\bar{T} - \mu| < ci) = cl$.

Note

We assume that the execution times form an independent sample from a normally distributed population, and use t distribution to estimate confidence interval.

Parameters

cl confidence level

reps number of measurements (should be > 1)

t array of reps measurement results

Returns

confidence interval

5.3.2.12 fupermod_data* fupermod_do_benchmark (fupermod_kernel * *kernel,* MPI_Comm *comm,* int *d,* int *sync*)

Performs one kernel

Parameters

kernel kernel

comm MPI communicator

d problem size

sync boolean switch to synchronise benchmarks.

Returns

data point

5.3.2.13 double* fupermod_do_benchmark_repitition (fupermod_kernel * *kernel*, MPI_Comm *comm*, int *d*, int *reps*, int *sync*)

Performs a number of benchmarks

Parameters

kernel kernel

comm MPI communicator

d problem size

reps number of repetitions

sync boolean switch to synchronise benchmarks.

Returns

array of times

5.3.2.14 int fupermod_comm_split (MPI_Comm *comm*, MPI_Comm * *newcomm*)

Split new comunicator for processes running on multicore node

Parameters

comm parent comunicator

newcomm child comunicator

Returns

Error status

5.3.2.15 fupermod_point fupermod_do_benchmark_ (fupermod_kernel * *kernel*, MPI_Comm *comm_global*, MPI_Comm *comm_node*, int *rank*, int *d*, fupermod_precision *precision*, fupermod_benchmark_mode *mode*)

Performs one benchmark

Parameters

kernel kernel

comm_global MPI global communicator

comm_node MPI communicator within a node

d problem size

precision precision

mode generating model mode

Returns

data point

5.3.2.16 FILE* fupermod_data_fopen (const char * *file_path*, fupermod_library * *routine*, const char *mode*, int *rank*, fupermod_benchmark_mode *benchmark_mode*)

Open model files for parallel I/O

Parameters

file_path external file path

routine routine

mode file r/w mode

rank process rank

benchmark_mode benchmark mode

Returns

file pointer

5.3.2.17 FILE* fupermod_data_file_open_host (const char * *file_path*, const char * *hostname*, fupermod_library * *routine*, const char *mode*)

Open model file for specified host to read or write

5.3.2.18 void trim_hostname (char * *hostname*)

Trims hostname to base

5.3.2.19 int read_machinefile (char * *machinefile*, char *** *hostnames*, int * *size*)

Reads hostnames from machinefile

Parameters

machinefile file holding hostnames

hostnames array of hostname strings (output of function)

size number of processors (output of function)

5.3.2.20 double fupermod_model_time (fupermod_model * *model*, const int *d*)

Returns estimated time

5.3.2.21 void fupermod_print_models (FILE * *stream*, const int *size*, fupermod_model ** *models*, const int *D*)

Prints models

5.3.2.22 fupermod_model* fupermod_model_interp_alloc (fupermod_data * *data*, int *D*, fupermod_kernel * *kernel*, fupermod_interp_type *type*, int *smooth*)

Allocates the model based on the Linear/Akima spline interpolation. Designed for models with few data points (partial models)

Parameters

data data points

D maximum problem size

kernel kernel

type type of interpolation

smooth smooth parameter for the Akima spline interpolation of full functional performance models (0 - no smoothing)

Returns

model

5.3.2.23 void fupermod_model_interp_free (fupermod_model * *model*)

Frees the model

5.3.2.24 void fupermod_model_multifit_free (fupermod_model * *model*)

Frees the model

5.3.2.25 fupermod_distr* fupermod_distr_alloc (int *size*, int *D*)

Allocates an equal distribution

5.3.2.26 void fupermod_distr_free (fupermod_distr * *distr*)

Frees the distribution

5.3.2.27 int fupermod_partition (int *D*, int *size*, fupermod_model ** *models*, int *iters_hybrj*, double *eps_hybrj*, fupermod_distr * *distr*)

Finds the optimal partition.

Solves a system of non-linear equations $f(\mathbf{x}) = 0$, where $f(\mathbf{x}) = \begin{cases} n - \sum_{i=1}^{p} x_i \\ \frac{x_i}{s_i(x_i)} - \frac{x_1}{s_1(x_1)} \quad 2 \leq i \leq p \end{cases}$ by the

least squares minimization of the residual of the system. Uses the HYBRJ algorithm (a modified version of Powell's Hybrid method) implemented in GSL. The iterative solution starts with $\mathbf{x_0}$ given by the distr argument. distr is the resulting distribution.

Parameters

D maximum problem size

size number of processors

models models

iters_hybrj maximum number of iterations for the HYBRJ algorithm

eps_hybrj maximum residual for the HYBRJ algorithm

distr optimal data distribution (input/output parameter)

5.3.2.28 int fupermod_partition_geom (int *D*, int *size*, fupermod_model ** *models*, fupermod_distr * *distr*)

Geometrical data partitioning

Parameters

D maximum problem size

size number of processors

models models

distr optimal data distribution (input/output parameter)

5.3.2.29 int fupermod_partition_cpm (int *D*, int *size*, double * *times*, fupermod_distr * *distr*)

Finds the constant performance model partition (cpm).

Parameters

D maximum problem size

size number of processors

times times

distr optimal data distribution (input/output parameter)

5.3.2.30 int fupermod_partition_dynamic (MPI_Comm *comm*, int *root*, int *iters*, int *eps*, fupermod_kernel * *kernel*, int *iters_hybrj*, double *eps_hybrj*, fupermod_distr * *distr*, fupermod_model ** *models*, char ** *emulate_hosts*, fupermod_library * *emulate_routine*)

Dynamically finds the optimal partition. Executes the kernel in parallel some times. Uses given distribution and models, updates them during the execution.

Parameters

comm MPI communicator

root rank of the root processor

iters number of dynamic partitoner iterations

eps error of dynamic partitioner

kernel kernel

iters_hybrj maximum number of iterations for the HYBRJ algorithm

eps_hybrj maximum residual for the HYBRJ algorithm

distr initial and then optimal data distribution (significant only at root)

models models, array of the communicator size (significant only at root)

emulate_hosts host names to emulate, used for reading correct host files.

emulate_routine only needed to get routine_name for reading FPM file.

5.3.2.31 int fupermod_balance (MPI_Comm *comm,* int *root,* fupermod_section *section,* struct timeval * *start,* int *D,* double *threshold,* fupermod_distr * *distr,* fupermod_model ** *models*)

Dynamically balances the load. Benchmarks the code section between two consecutive calls of this function. The kernel argument is used for information about the complexity of the code. Uses given distribution and models, updates them during the execution.

Parameters

comm MPI communicator

root rank of the root processor

section code section to balance

start start time

D maximum problem size

threshold Relative time difference above which the computation is considered balanced. Between 0 and 1.

distr initial and then optimal data distribution (significant only at root)

models models, array of the communicator size (significant only at root)

5.3.2.32 void fupermod_balance_init (MPI_Comm *comm,* int *root,* fupermod_algorithm *algorithm,* int *D,* char * *info*)

Initialises fupermod_balance, to be called before iterative loop. Sets header of time data file and measures total time of routine. Called by all processors.

Parameters

comm MPI communicator

root rank of the root processor

algorithm balancing algoritm to use (ie. pfpm, cpm or manual)

D maximum problem size

info string containing extra routine specific info

5.3.2.33 void fupermod_balance_finalise (int *size*, fupermod_model ** *models*)

Finalise balancing algorithm. Called only by root after iterative loop.

5.4 Functional performance model of heterogeneous multiprocessors and multicores

Classes

- class fupermod_multi_data
- class fupermod_multi_model

5.5 2D routines API

Classes

- struct fupermod_2d_kernel
- struct fupermod_2d_routine

Typedefs

- typedef double(* fupermod_2d_complexity)(int m, int n)
- typedef int(* fupermod_2d_complexity_derivatives)(int m, int n, double *dm, double *dn)
- typedef int(* fupermod_2d_x0)(int M, int N, int size, double *m, double *n)
- typedef int(* fupermod_2d_f)(int size, double *m, double *n, int M, int N, double *f)
- typedef int(* fupermod_2d_df)(int size, double *m, double *n, int M, int N, double *J)
- typedef int(* fupermod_2d_remainders)(int M, int N, int size, double *x, double *y, int *new_m, int *new_n)
- typedef int(* fupermod_2d_initialize)(MPI_Comm comm, int m, int n)

5.5.1 Detailed Description

Each routine should be implemented in a shared library as an instance of the Routines API structure. The fupermod tools will search for a symbol of this instance.

5.5.2 Typedef Documentation

5.5.2.1 typedef double(* fupermod_2d_complexity)(int m, int n)

2D routine/kernel complexity

Parameters

m problem size

n problem size

Returns

complexity

5.5.2.2 typedef int(∗ fupermod_2d_complexity_derivatives)(int m, int n, double ∗dm, double ∗dn)

2D kernel complexity derivatives

Parameters

m problem size

n problem size

Returns

complexity

5.5.2.3 typedef int(∗ fupermod_2d_x0)(int M, int N, int size, double ∗m, double ∗n)

Initial 2D partition

Parameters

M total problem size

N total problem size

size number of processors

m problem sizes

n problem sizes

Returns

return code

5.5.2.4 typedef int(∗ fupermod_2d_f)(int size, double ∗m, double ∗n, int M, int N, double ∗f)

2D partition function

Parameters

size number of processors

m problem sizes

n problem sizes

M total problem size

N total problem size

f function (array of size + 1)

Returns

return code

5.5.2.5 typedef int(∗ fupermod_2d_df)(int size, double ∗m, double ∗n, int M, int N, double ∗J)

Jacobian for 2D partition function

Parameters

size number of processors

m problem sizes

n problem sizes

M total problem size

N total problem size

J Jacobian (array of (size + 1) ∗ (2 ∗ size)), x = (m, n)

Returns

return code

5.5.2.6 typedef int(∗ fupermod_2d_remainders)(int M, int N, int size, double ∗x, double ∗y, int ∗new_m, int ∗new_n)

2D distribution of remainders

Parameters

M total problem size

N total problem size

size number of processors

m problem sizes

n problem sizes

new_m problem sizes

new_n problem sizes

Returns

return code

5.5.2.7 typedef int(∗ fupermod_2d_initialize)(MPI_Comm comm, int m, int n)

Initializes 2D routine/kernel. Allocates data for benchmarking, which are stored in global variables.

Parameters

comm MPI communicator

m problem size

n problem size

Returns

return code

5.6 2D functional performance model based on scattered points

Classes

- struct fupermod_2d_data
- struct fupermod_2d_model
- struct fupermod_2d_distr

Functions

- fupermod_2d_data * fupermod_2d_data_alloc ()
- void fupermod_2d_data_free (fupermod_2d_data *data)
- fupermod_2d_data * fupermod_2d_data_read (FILE *stream)
- void fupermod_2d_data_write (FILE *stream, fupermod_2d_data *data)
- int fupermod_2d_data_index (fupermod_2d_data *data, int m, int n)
- int fupermod_2d_data_add (fupermod_2d_data *data, int m, int n, double t)
- fupermod_2d_data * fupermod_2d_data_benchmark (fupermod_2d_kernel *kernel, int M, int N, int m_stride, int n_stride)
- fupermod_2d_model * fupermod_2d_model_build (fupermod_2d_data *data, fupermod_2d_kernel *kernel, int iters)
- void fupermod_2d_model_free (fupermod_2d_model *model)
- fupermod_2d_distr * fupermod_2d_distr_alloc (int size)
- void fupermod_2d_distr_free (fupermod_2d_distr *distr)
- int fupermod_2d_partition (int M, int N, int size, fupermod_2d_model **models, int iters_hybrj, double eps_hybrj, fupermod_2d_distr *distr)

Variables

- int fupermod_verbose

5.6.1 Function Documentation

5.6.1.1 fupermod_2d_data* fupermod_2d_data_alloc ( )

Allocates the data

5.6.1.2 void fupermod_2d_data_free (fupermod_2d_data * *data*)

Frees the data

5.6.1.3 fupermod_2d_data* fupermod_2d_data_read (FILE * *stream*)

Reads data from a file

5.6.1.4 void fupermod_2d_data_write (FILE * *stream*, fupermod_2d_data * *data*)

Writes data to a file

5.6.1.5 int fupermod_2d_data_index (fupermod_2d_data * *data*, int *m*, int *n*)

Returns an index of the data point or -1

5.6.1.6 int fupermod_2d_data_add (fupermod_2d_data * *data*, int *m*, int *n*, double *t*)

Adds the data point

Parameters

data data points

m problem size

n problem size

t time

Returns

index or -1

5.6.1.7 fupermod_2d_data* fupermod_2d_data_benchmark (fupermod_2d_kernel * *kernel*, int *M*, int *N*, int *m_stride*, int *n_stride*)

Benchmark

5.6.1.8 fupermod_2d_model* fupermod_2d_model_build (fupermod_2d_data * *data*, fupermod_2d_kernel * *kernel*, int *iters*)

Builds the model based on least squares 8-level cubic C^2 B-spline approximation

Parameters

data data points

kernel kernel

iters maximum number of iterations in conjugate gradient relaxation

Returns

model approximation

5.6.1.9 void fupermod_2d_model_free (fupermod_2d_model * *model*)

Frees the model

5.6.1.10 fupermod_2d_distr* fupermod_2d_distr_alloc (int *size*)

Allocates the distribution

5.6.1.11 void fupermod_2d_distr_free (fupermod_2d_distr * *distr*)

Frees the distribution

5.6.1.12 int fupermod_2d_partition (int *M*, int *N*, int *size*, fupermod_2d_model ** *models*, int *iters_hybrj*, double *eps_hybrj*, fupermod_2d_distr * *distr*)

Finds the optimal partition. Solves a system of non-linear equations $f(\mathbf{x}, \mathbf{y}) = 0$, where $f(\mathbf{x}, \mathbf{y}) = \begin{cases} g_j(\mathbf{x}, \mathbf{y}) & 1 \leq j \leq p+1 \\ \dfrac{c(x_i, y_i)}{s_i(x_i, y_i)} - \dfrac{c(x_1, y_1)}{s_1(x_1, y_1)} & 2 \leq i \leq p \end{cases}$ by the least squares minimization of the residual of the system. Uses the HYBRJ algorithm (a modified version of Powell's Hybrid method) implemented in GSL. $g(\mathbf{x}, \mathbf{y})$ provided by the routine. The iterative solution starts with $\mathbf{x_0}, \mathbf{y_0}$ provided by the routine.

Parameters

M maximum problem size

N maximum problem size

size number of processors

models models

iters_hybrj maximum number of iterations for the HYBRJ algorithm

eps_hybrj maximum residual for the HYBRJ algorithm

distr optimal data distribution

5.6.2 Variable Documentation

5.6.2.1 int fupermod_verbose

Verbose level (default: 0 - none)

6 Class Documentation

6.1 fupermod_2d_data Struct Reference

Public Attributes

- int count
- int * m
- int * n
- double * t

6.1.1 Detailed Description

2D scattered points. Life cycle: created before model, destroyed after todo: sorted?

6.1.2 Member Data Documentation

6.1.2.1 int fupermod_2d_data::count

Number of points

6.1.2.2 int* fupermod_2d_data::m

Problem sizes

6.1.2.3 int* fupermod_2d_data::n

Problems sizes

6.1.2.4 double* fupermod_2d_data::t

Times

The documentation for this struct was generated from the following file:

- src/fupermod_2d.h

6.2 fupermod_2d_distr Struct Reference

Public Attributes

- int size
- int * m
- int * n
- double * t
- double * s

6.2.1 Detailed Description

2D data distribution

6.2.2 Member Data Documentation

6.2.2.1 int fupermod_2d_distr::size

Number of processors

6.2.2.2 int* fupermod_2d_distr::m

Problem sizes

6.2.2.3 int* fupermod_2d_distr::n

Problem sizes

6.2.2.4 double* fupermod_2d_distr::t

Times

6.2.2.5 double* fupermod_2d_distr::s

Speeds

The documentation for this struct was generated from the following file:

- src/fupermod_2d.h

6.3 fupermod_2d_kernel Struct Reference

Public Attributes

- fupermod_2d_complexity complexity
- fupermod_2d_complexity_derivatives complexity_derivatives
- fupermod_2d_x0 x0
- fupermod_2d_f f
- fupermod_2d_df df
- fupermod_2d_remainders remainders
- fupermod_2d_initialize initialize
- fupermod_execute execute
- fupermod_finalize finalize

6.3.1 Detailed Description

2D kernel

6.3.2 Member Data Documentation

6.3.2.1 fupermod_2d_complexity fupermod_2d_kernel::complexity

Complexity

6.3.2.2 fupermod_2d_complexity_derivatives fupermod_2d_kernel::complexity_derivatives

Complexity derivatives

6.3.2.3 fupermod_2d_x0 fupermod_2d_kernel::x0

Initial distribution

6.3.2.4 fupermod_2d_f fupermod_2d_kernel::f

2D partition

6.3.2.5 fupermod_2d_df fupermod_2d_kernel::df

Jacobian for 2D partition

6.3.2.6 fupermod_2d_remainders fupermod_2d_kernel::remainders

Distribution of remainders

6.3.2.7 fupermod_2d_initialize fupermod_2d_kernel::initialize

Initialization

6.3.2.8 fupermod_execute fupermod_2d_kernel::execute

Execution

6.3.2.9 fupermod_finalize fupermod_2d_kernel::finalize

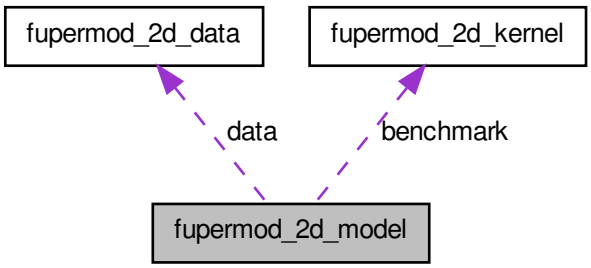
Finalization

The documentation for this struct was generated from the following file:

- src/fupermod_2d_routine.h

6.4 fupermod_2d_model Struct Reference

Collaboration diagram for fupermod_2d_model:



Public Attributes

- fupermod_2d_data * data
- fupermod_2d_kernel * benchmark
- double(* evaluate)(struct fupermod_2d_model *model, double x, double y)
- int(* evaluate_derivatives)(struct fupermod_2d_model *model, double x, double y, double *dx, double *dy)

6.4.1 Detailed Description

2D model based on scattered points

6.4.2 Member Data Documentation

6.4.2.1 fupermod_2d_data* fupermod_2d_model::data

Data

6.4.2.2 fupermod_2d_kernel* fupermod_2d_model::benchmark

Benchmark

6.4.2.3 double(* fupermod_2d_model::evaluate)(struct fupermod_2d_model *model, double x, double y)

Evaluates the model. Must return positive values.

6.4.2.4 int(* fupermod_2d_model::evaluate_derivatives)(struct fupermod_2d_model *model, double x, double y, double *dx, double *dy)

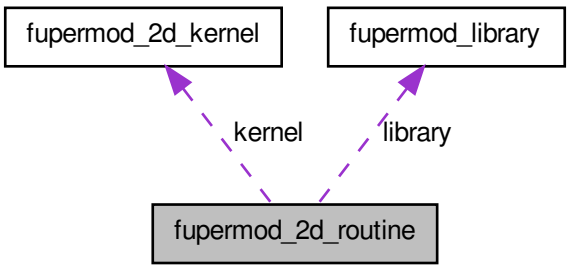
Evaluates the derivative

The documentation for this struct was generated from the following file:

- src/fupermod_2d.h

6.5 fupermod_2d_routine Struct Reference

Collaboration diagram for fupermod_2d_routine:



Public Attributes

- fupermod_library library
- fupermod_2d_complexity complexity
- fupermod_2d_initialize initialize
- fupermod_execute execute
- fupermod_finalize finalize
- fupermod_2d_kernel kernel

6.5.1 Detailed Description

2D routine

6.5.2 Member Data Documentation

6.5.2.1 fupermod_library fupermod_2d_routine::library

Library

6.5.2.2 fupermod_2d_complexity fupermod_2d_routine::complexity

Complexity

6.5.2.3 fupermod_2d_initialize fupermod_2d_routine::initialize

Initialization

6.5.2.4 fupermod_execute fupermod_2d_routine::execute

Execution

6.5.2.5 fupermod_finalize fupermod_2d_routine::finalize

Finalization

6.5.2.6 fupermod_2d_kernel fupermod_2d_routine::kernel

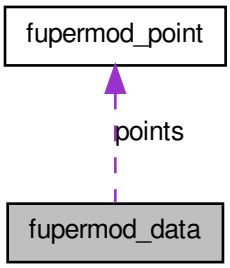
Kernel. Its fields may be empty, if the routine is iterative - there is no need in benchmarking the kernel

The documentation for this struct was generated from the following file:

- src/fupermod_2d_routine.h

6.6 fupermod_data Struct Reference

Collaboration diagram for fupermod_data:



Public Attributes

- int count
- fupermod_point * points

6.6.1 Detailed Description

Data points sorted by problem size. Problem sizes must be positive. Life cycle: created before model, destroyed after. To find a data point, use fupermod_data_index. To add a data point, use fupermod_data_add.

6.6.2 Member Data Documentation

6.6.2.1 int fupermod_data::count

Number of points

6.6.2.2 fupermod_point* fupermod_data::points

Points

The documentation for this struct was generated from the following file:

- src/fupermod_data.h

6.7 fupermod_distr Struct Reference

Public Attributes

- int size

- int D
- int * d
- double * t
- double * s

6.7.1 Detailed Description

Data distribution

6.7.2 Member Data Documentation

6.7.2.1 int fupermod_distr::size

Number of processors

6.7.2.2 int fupermod_distr::D

Total problem size

6.7.2.3 int* fupermod_distr::d

Problem sizes

6.7.2.4 double* fupermod_distr::t

Times

6.7.2.5 double* fupermod_distr::s

Speeds

The documentation for this struct was generated from the following file:

- src/fupermod_distr.h

6.8 fupermod_kernel Struct Reference

Public Attributes

- fupermod_complexity complexity
- fupermod_initialize initialize
- fupermod_execute execute
- fupermod_finalize finalize

6.8.1 Detailed Description

Benchmark

6.8.2 Member Data Documentation

6.8.2.1 fupermod_complexity fupermod_kernel::complexity

Complexity

6.8.2.2 fupermod_initialize fupermod_kernel::initialize

Initialization

6.8.2.3 fupermod_execute fupermod_kernel::execute

Execution. No communications, code should be serial

6.8.2.4 fupermod_finalize fupermod_kernel::finalize

Finalization

The documentation for this struct was generated from the following file:

- src/fupermod_routine.h

6.9 fupermod_library Struct Reference

Public Attributes

- int dim
- const char * name
- int(* conf)(FILE *stream)
- int(* load)(MPI_Comm comm, char *subopts)
- int(* unload)(MPI_Comm comm)

6.9.1 Detailed Description

Library. The fupermod tools search for an instance of this structure. Then, it is type-casted to either Routines API or 2D routines API.

See also

FUPERMOD_ROUTINE_INSTANCE

6.9.2 Member Data Documentation

6.9.2.1 int fupermod_library::dim

Routine dimension (1 or 2)

6.9.2.2 const char* fupermod_library::name

Routine name

6.9.2.3 int(∗ fupermod_library::conf)(FILE ∗stream)

Prints information about the configuration. Each line starts with '#'

6.9.2.4 int(∗ fupermod_library::load)(MPI_Comm comm, char ∗subopts)

Loads library. Parses suboptions at root and broadcasts the routines settings in communicator. Some tools are non-MPI, therefore MPI_COMM_NULL should be handled.

Parameters

comm MPI communicator

subopts suboptions

Returns

return code

6.9.2.5 int(∗ fupermod_library::unload)(MPI_Comm comm)

Unloads library. Some tools are non-MPI, therefore MPI_COMM_NULL should be handled.

Parameters

comm MPI communicator

Returns

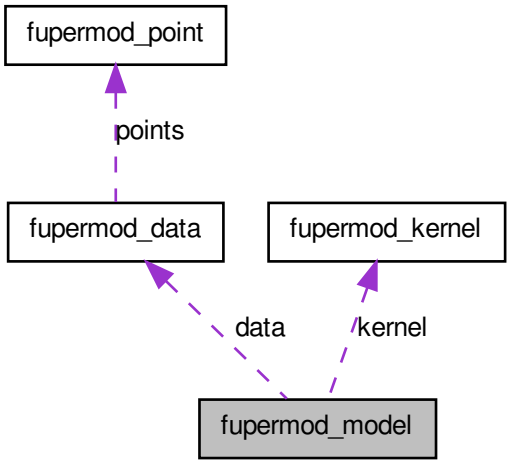
return code

The documentation for this struct was generated from the following file:

- src/fupermod_routine.h

6.10 fupermod_model Struct Reference

Collaboration diagram for fupermod_model:



Public Attributes

- fupermod_data * data
- int D
- fupermod_kernel * kernel
- double(* f)(struct fupermod_model *model, double x)
- double(* df)(struct fupermod_model *model, double x)
- void(* fdf)(struct fupermod_model *model, double x, double *f, double *df)
- void(* update)(struct fupermod_model *model)

6.10.1 Detailed Description

Model: approximation of speed (based on data points and kernel complexity). Must be defined at [0; D]. Abstract data structure. Encapsulates and updates the data points provided by fupermod_data. Ensure data integrity (data points <-> model): after a new data point is added by fupermod_data_add, the model must be updated by update.

6.10.2 Member Data Documentation

6.10.2.1 fupermod_data* fupermod_model::data

Data

6.10.2.2 int fupermod_model::D

Maximum problem size

6.10.2.3 fupermod_kernel* fupermod_model::kernel

Benchmark

6.10.2.4 double(* fupermod_model::f)(struct fupermod_model *model, double x)

Evaluates the function. Must return positive values.

6.10.2.5 double(* fupermod_model::df)(struct fupermod_model *model, double x)

Evaluates the derivative

6.10.2.6 void(* fupermod_model::fdf)(struct fupermod_model *model, double x, double *f, double *df)

Evaluates both

6.10.2.7 void(* fupermod_model::update)(struct fupermod_model *model)

Updates the model

The documentation for this struct was generated from the following file:

- src/fupermod_model.h

6.11 fupermod_multi_data Class Reference

6.11.1 Detailed Description

Data

The documentation for this class was generated from the following file:

- src/fupermod_multi.hpp

6.12 fupermod_multi_model Class Reference

6.12.1 Detailed Description

Model

The documentation for this class was generated from the following file:

- src/fupermod_multi.hpp

6.13 fupermod_point Struct Reference

Public Attributes

- int d
- double t
- int reps
- double ci

6.13.1 Detailed Description

Data point

6.13.2 Member Data Documentation

6.13.2.1 int fupermod_point::d

Problem size

6.13.2.2 double fupermod_point::t

Time

6.13.2.3 int fupermod_point::reps

Number of repetitions the kernel has actually taken

6.13.2.4 double fupermod_point::ci

Confidence interval, $|\bar{T} - \mu| < ci$. The fupermod_ci function estimates confidence interval, using t distribution.

The documentation for this struct was generated from the following file:

- src/fupermod_data.h

6.14 fupermod_precision Struct Reference

Public Attributes

- int reps_min
- int reps_max
- double cl
- double eps

6.14.1 Detailed Description

Precision of measurement. Used as an input argument of benchmark functions. To provide reliable results, the computation experiments in each benchmark are repeated either fixed or variable number of times. This data structure allows the user to control the accuracy and efficiency of benchmarking.

- Assigning to reps_min and reps_max the same values results in the fixed number of repetitions of the computation operation, with the cl and eps arguments being ignored (this allows the user to control the efficiency of benchmarking).
- If reps_min $<$ reps_max, the experiments are repeated until a confidence interval, fupermod_point::ci, found with the confidence level, cl = $Pr(|\bar{T} - \mu| < ci)$, satisfies $\frac{ci}{\bar{T}} <$ eps, or the number of repetitions reaches its maximum, reps_max (this allows the user to control the accuracy of benchmarking).

6.14.2 Member Data Documentation

6.14.2.1 int fupermod_precision::reps_min

Minimum number of repetitions

6.14.2.2 int fupermod_precision::reps_max

Maximum number of repetitions

6.14.2.3 double fupermod_precision::cl

Confidence level

6.14.2.4 double fupermod_precision::eps

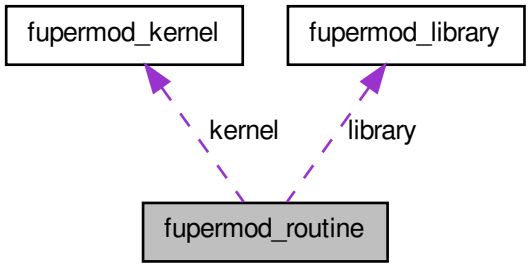
Relative error

The documentation for this struct was generated from the following file:

- src/fupermod_benchmark.h

6.15 fupermod_routine Struct Reference

Collaboration diagram for fupermod_routine:



Public Attributes

- fupermod_library library
- fupermod_complexity complexity
- fupermod_initialize initialize
- fupermod_execute execute
- fupermod_finalize finalize
- fupermod_kernel kernel

6.15.1 Detailed Description

Routine

6.15.2 Member Data Documentation

6.15.2.1 fupermod_library fupermod_routine::library

Library

6.15.2.2 fupermod_complexity fupermod_routine::complexity

Complexity

6.15.2.3 fupermod_initialize fupermod_routine::initialize

Initialization

6.15.2.4 fupermod_execute fupermod_routine::execute

Execution

6.15.2.5 fupermod_finalize fupermod_routine::finalize

Finalization

6.15.2.6 fupermod_kernel fupermod_routine::kernel

Kernel. Its fields may be empty, if the routine is iterative - there is no need in benchmarking the kernel The documentation for this struct was generated from the following file:

- src/fupermod_routine.h

Index

2D functional performance model based on scattered points, 29
2D routines API, 26

benchmark
    fupermod_2d_model, 35

ci
    fupermod_point, 43
cl
    fupermod_precision, 44
complexity
    fupermod_2d_kernel, 33
    fupermod_2d_routine, 36
    fupermod_kernel, 39
    fupermod_routine, 45
complexity_derivatives
    fupermod_2d_kernel, 33
conf
    fupermod_library, 39
constant1
    fupermod, 18
constant2
    fupermod, 18
count
    fupermod_2d_data, 31
    fupermod_data, 37

D
    fupermod_distr, 38
    fupermod_model, 41
d
    fupermod_distr, 38
    fupermod_point, 43
data
    fupermod_2d_model, 35
    fupermod_model, 41
df
    fupermod_2d_kernel, 33
    fupermod_model, 42
dim
    fupermod_library, 39

eps
    fupermod_precision, 44
evaluate
    fupermod_2d_model, 35
evaluate_derivatives
    fupermod_2d_model, 35
execute
    fupermod_2d_kernel, 34
    fupermod_2d_routine, 36
    fupermod_kernel, 39
    fupermod_routine, 45

f
    fupermod_2d_kernel, 33
    fupermod_model, 42
fdf
    fupermod_model, 42
finalize
    fupermod_2d_kernel, 34
    fupermod_2d_routine, 36
    fupermod_kernel, 39
    fupermod_routine, 46
full
    fupermod, 18
Functional performance model, 17
Functional performance model of heterogeneous multiprocessors and multicores, 26
fupermod
    constant1, 18
    constant2, 18
    full, 18
    fupermod_interp_akima, 18
    fupermod_interp_linear, 18
    fupermod_algorithm, 18
    fupermod_balance, 25
    fupermod_balance_finalise, 25
    fupermod_balance_init, 25
    fupermod_benchmark_mode, 18
    fupermod_ci, 20
    fupermod_comm_split, 21
    fupermod_data_add, 19
    fupermod_data_alloc, 19
    fupermod_data_file_open_host, 22
    fupermod_data_fopen, 21
    fupermod_data_free, 19
    fupermod_data_index, 19
    fupermod_data_read, 19
    fupermod_data_time, 19
    fupermod_data_write, 19
    fupermod_data_write_head, 19
    fupermod_distr_alloc, 23
    fupermod_distr_free, 23
    fupermod_do_benchmark, 20
    fupermod_do_benchmark_, 21
    fupermod_do_benchmark_repitition, 20
    fupermod_interp_type, 18
    fupermod_model_interp_alloc, 22
    fupermod_model_interp_free, 23
    fupermod_model_multifit_free, 23

fupermod_model_time, 22
fupermod_partition, 23
fupermod_partition_cpm, 24
fupermod_partition_dynamic, 24
fupermod_partition_geom, 24
fupermod_point_read, 19
fupermod_point_write, 19
fupermod_print_models, 22
fupermod_section, 18
homogeneous, 18
manual, 18
mc_mc, 18
mc_mp, 18
partial, 18
read_machinefile, 22
sc_mp, 18
trim_hostname, 22
fupermod_interp_akima
fupermod, 18
fupermod_interp_linear
fupermod, 18
fupermod_2d
fupermod_2d_data_add, 30
fupermod_2d_data_alloc, 29
fupermod_2d_data_benchmark, 30
fupermod_2d_data_free, 29
fupermod_2d_data_index, 29
fupermod_2d_data_read, 29
fupermod_2d_data_write, 29
fupermod_2d_distr_alloc, 30
fupermod_2d_distr_free, 30
fupermod_2d_model_build, 30
fupermod_2d_model_free, 30
fupermod_2d_partition, 30
fupermod_verbose, 31
fupermod_2d_complexity
fupermod_2d_routine, 26
fupermod_2d_complexity_derivatives
fupermod_2d_routine, 26
fupermod_2d_data, 31
count, 31
m, 31
n, 32
t, 32
fupermod_2d_data_add
fupermod_2d, 30
fupermod_2d_data_alloc
fupermod_2d, 29
fupermod_2d_data_benchmark
fupermod_2d, 30
fupermod_2d_data_free
fupermod_2d, 29
fupermod_2d_data_index
fupermod_2d, 29
fupermod_2d_data_read
fupermod_2d, 29
fupermod_2d_data_write
fupermod_2d, 29
fupermod_2d_df
fupermod_2d_routine, 27
fupermod_2d_distr, 32
m, 32
n, 32
s, 32
size, 32
t, 32
fupermod_2d_distr_alloc
fupermod_2d, 30
fupermod_2d_distr_free
fupermod_2d, 30
fupermod_2d_f
fupermod_2d_routine, 27
fupermod_2d_initialize
fupermod_2d_routine, 28
fupermod_2d_kernel, 33
complexity, 33
complexity_derivatives, 33
df, 33
execute, 34
f, 33
finalize, 34
initialize, 34
remainders, 33
x0, 33
fupermod_2d_model, 34
benchmark, 35
data, 35
evaluate, 35
evaluate_derivatives, 35
fupermod_2d_model_build
fupermod_2d, 30
fupermod_2d_model_free
fupermod_2d, 30
fupermod_2d_partition
fupermod_2d, 30
fupermod_2d_remainders
fupermod_2d_routine, 28
fupermod_2d_routine, 35
complexity, 36
execute, 36
finalize, 36
fupermod_2d_complexity, 26
fupermod_2d_complexity_derivatives, 26
fupermod_2d_df, 27
fupermod_2d_f, 27
fupermod_2d_initialize, 28
fupermod_2d_remainders, 28
fupermod_2d_x0, 27

initialize, 36
kernel, 36
library, 36
fupermod_2d_x0
fupermod_2d_routine, 27
fupermod_algorithm
fupermod, 18
fupermod_balance
fupermod, 25
fupermod_balance_finalise
fupermod, 25
fupermod_balance_init
fupermod, 25
fupermod_benchmark_mode
fupermod, 18
fupermod_ci
fupermod, 20
fupermod_comm_split
fupermod, 21
fupermod_complexity
fupermod_routine, 16
fupermod_data, 37
count, 37
points, 37
fupermod_data_add
fupermod, 19
fupermod_data_alloc
fupermod, 19
fupermod_data_file_open_host
fupermod, 22
fupermod_data_fopen
fupermod, 21
fupermod_data_free
fupermod, 19
fupermod_data_index
fupermod, 19
fupermod_data_read
fupermod, 19
fupermod_data_time
fupermod, 19
fupermod_data_write
fupermod, 19
fupermod_data_write_head
fupermod, 19
fupermod_distr, 37
D, 38
d, 38
s, 38
size, 38
t, 38
fupermod_distr_alloc
fupermod, 23
fupermod_distr_free
fupermod, 23
fupermod_do_benchmark
fupermod, 20
fupermod_do_benchmark_
fupermod, 21
fupermod_do_benchmark_repitition
fupermod, 20
fupermod_execute
fupermod_routine, 16
fupermod_finalize
fupermod_routine, 16
fupermod_initialize
fupermod_routine, 16
fupermod_interp_type
fupermod, 18
fupermod_kernel, 38
complexity, 39
execute, 39
finalize, 39
initialize, 39
fupermod_library, 39
conf, 39
dim, 39
load, 40
name, 39
unload, 40
fupermod_model, 41
D, 41
data, 41
df, 42
f, 42
fdf, 42
kernel, 42
update, 42
fupermod_model_interp_alloc
fupermod, 22
fupermod_model_interp_free
fupermod, 23
fupermod_model_multifit_free
fupermod, 23
fupermod_model_time
fupermod, 22
fupermod_multi_data, 42
fupermod_multi_model, 42
fupermod_partition
fupermod, 23
fupermod_partition_cpm
fupermod, 24
fupermod_partition_dynamic
fupermod, 24
fupermod_partition_geom
fupermod, 24
fupermod_point, 43
ci, 43
d, 43

reps, 43
t, 43
fupermod_point_read
fupermod, 19
fupermod_point_write
fupermod, 19
fupermod_precision, 43
cl, 44
eps, 44
reps_max, 44
reps_min, 44
fupermod_print_models
fupermod, 22
fupermod_routine, 45
complexity, 45
execute, 45
finalize, 46
fupermod_complexity, 16
fupermod_execute, 16
fupermod_finalize, 16
fupermod_initialize, 16
FUPERMOD_ROUTINE_INSTANCE, 15
FUPERMOD_SUCCESS, 15
initialize, 45
kernel, 46
library, 45
FUPERMOD_ROUTINE_INSTANCE
fupermod_routine, 15
fupermod_section
fupermod, 18
FUPERMOD_SUCCESS
fupermod_routine, 15
fupermod_utilities
fupermod_verbose, 15
fupermod_verbose
fupermod_2d, 31
fupermod_utilities, 15

homogeneous
fupermod, 18

initialize
fupermod_2d_kernel, 34
fupermod_2d_routine, 36
fupermod_kernel, 39
fupermod_routine, 45

kernel
fupermod_2d_routine, 36
fupermod_model, 42
fupermod_routine, 46

library
fupermod_2d_routine, 36
fupermod_routine, 45
load
fupermod_library, 40

m
fupermod_2d_data, 31
fupermod_2d_distr, 32
manual
fupermod, 18
mc_mc
fupermod, 18
mc_mp
fupermod, 18

n
fupermod_2d_data, 32
fupermod_2d_distr, 32
name
fupermod_library, 39

partial
fupermod, 18
points
fupermod_data, 37

read_machinefile
fupermod, 22
remainders
fupermod_2d_kernel, 33
reps
fupermod_point, 43
reps_max
fupermod_precision, 44
reps_min
fupermod_precision, 44
Routines API, 15

s
fupermod_2d_distr, 32
fupermod_distr, 38
sc_mp
fupermod, 18
size
fupermod_2d_distr, 32
fupermod_distr, 38

t
fupermod_2d_data, 32
fupermod_2d_distr, 32
fupermod_distr, 38
fupermod_point, 43
trim_hostname
fupermod, 22

unload

fupermod_library, 40
update
fupermod_model, 42
Utilities, 15

x0
fupermod_2d_kernel, 33