ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

**РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ**

по дисциплине “Параллельное программирование”

на тему

**Разработка библиотеки отказоустойчивости для вычислительных систем с массовым параллелизмом**

|  |  |
| --- | --- |
| Выполнил студент | Марков В.А. |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | МГ-165 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | д.т.н. М.Г. Курносов |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2016

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#__RefHeading___Toc337081083) 3

[1. Контрольные точки на уровне пользователя](#__RefHeading___Toc337081084)…..………………………………...4

2. Алгоритм выполнения программы при использовании контрольных точек ………………………..……………………………………………………………….5

3. Реализация техники контрольных точек на уровне пользователя …………….6

[ЗАКЛЮЧЕНИЕ 8](#__RefHeading___Toc337081086)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#__RefHeading___Toc337081087) 9

[ПРИЛОЖЕНИЕ 1](#__RefHeading___Toc337081088)0

ВВЕДЕНИЕ

Для существующих вычислительных систем петафлопсного уровня производительности и проектируемых систем экзафлопсного уровня существенное значение имеет показатель среднего времени безотказной работы. Разработка надежных высокопроизводительных систем на аппаратном уровне является трудноразрешимой задачей. В связи с этим, для организации вычислений на современных суперкомпьютерах необходимо развитие новых отказоустойчивых технологий, позволяющих с помощью программных решений корректно продолжать вычисления даже при отказе части оборудования [1].

Обеспечение отказоустойчивости на программном уроне классифицируется следующим образом:

* Протоколы репликации (replication protocols);
* Протоколы восстановления, основанных на откате (rollback recovery protocols);
* Протоколы само стабилизации (self stabilizing protocols);
* Логирование сообщений (message logging): оптимистичное, пессимистичное, обычное;
* Глобальные/локальные точки сохранения.

1. Контрольные точки на уровне пользователя

Техника контрольных точек (КТ) уровня пользователя (user-level checkpointing) задействует библиотеку (checkpointing library), предоставляя гибкий механизм реализации, в отличии от ограниченных возможностей КТ уровня ядра ОС.

В случае реализации отказоустойчивости на уровне пользователя, в контрольную точку входит только то, что явно укажет прикладной программист. В идеале, только те данные, которые необходимы для восстановления утерянной в результате сбоя информации. На уровне пользователя также можно контролировать время и частоту создания контрольных точек при выполнении программы, выбирая наиболее удобные моменты для их создания. Накладные расходы могут быть значительно уменьшены, однако, потребуется дополнительная работа прикладного программиста для реализации отказоустойчивости в приложении.

Преимущество такого подхода состоит в том, что КТ могут быть восстановлены на любом типе архитектуры. Тем не менее, КТ на уровне приложений требуют доступ к исходному коду пользователя и не поддерживают произвольное расположение контрольных точек. Таким образом, код пользователя должен быть инструментирован контрольным точками (часто вставляемых вручную программистом). В отличие от kernel-level checkpointing, user-level checkpointing не сигнализируются сигналами. Для того, чтобы контрольная точка сохранилась, ход выполнения программы должен достичь местоположения контрольной точки.

2. Алгоритм выполнения программы при использовании контрольных точек



Рисунок 1 – Алгоритм работы КТ

3. Реализация техники контрольных точек на уровне пользователя

int main()

{

// code . . .

int idx = get\_checkpoint\_index();

switch(idx)

{

Case 0: goto label\_0;

Case 1: goto label\_1;

}

// code . . .

label\_0:

func\_1();

label\_1:

func\_2();

// code . . .

return 0;

}

Рисунок 2 – Пример реализации

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Global variables \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern double cpl\_start\_time;

extern void \*\*cpl\_checkpoint\_table;

extern int cpl\_size;

extern int cpl\_time;

extern int cpl\_counter;

enum {

CPL\_CHECKPOINT\_MODE = 0,

CPL\_RECOVERY\_MODE = 1

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Initializing checkpoint library macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Decription:

\* size - checkpoints numbers

\* time - in seconds for timer

\* func - handler function

\*/

#define CPL\_INIT(size, time, func) \

signal(SIGALRM, func); \

cpl\_size = time; \

cpl\_counter = 0; \

cpl\_size = size; \

cpl\_start\_time = wtime\_(); \

cpl\_checkpoint\_table = init\_table\_(cpl\_size); \

//#define CPL\_DEINIT() deinit\_table\_();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Declaration checkpoint macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Description:

\* name - checkpoint\_one, checkpoint\_two, etc

\* - phase\_one, phase\_two, etc

\* - one, two, three, etc

\*/

#define CPL\_DECLARATE\_CHECKPOINT(name) \

cpl\_checkpoint\_table[cpl\_counter++] = name; \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Control flow-macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_GO\_TO\_CHECKPOINT(idx) \

goto \*cpl\_checkpoint\_table[idx]; \

#define CPL\_SET\_CHECKPOINT(checkpoint\_name) \

checkpoint\_name : \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Checkpoint-save macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_FILE\_OPEN(file, phase) \

open\_snapshot\_file\_(file, phase); \

#define CPL\_FILE\_CLOSE(file) \

close\_snapshot\_file\_(file); \

#define CPL\_SAVE\_SNAPSHOT(file, data, n, type) \

write\_to\_snapshot\_(file, data, n, type); \

#define CPL\_GET\_SNAPSHOT(snapshot) \

get\_last\_snapshot\_(snapshot); \

#define CPL\_SAVE\_STATE(checkpoint, user\_save\_callback) \

user\_save\_callback(get\_checkpoint\_idx\_by\_name\_(cpl\_checkpoint\_table, cpl\_size, checkpoint));\

Рисунок 3 – Реализованные макросы

ЗАКЛЮЧЕНИЕ

В результате выполнения расчетно-графического задания спроектирована и разработана библиотека отказоустойчивости для вычислительных систем с массовым параллелизмом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек URL: http://cyberleninka.ru/article/n/obespechenie-otkazoustoychivosti-vysokoproizvoditelnyh-vychisleniy-s-pomoschyu-lokalnyh-kontrolnyh-tochek
2. Моделирование отказов в высокопроизводительных вычислительных системах в рамках стандарта MPI и его расширения ULFM URL: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=vyurv&paperid=1&option\_lang=rus
3. Оптимизация времени создания и объёма контрольных точек восстановления параллельных программ URL: http://vestnik.sibsutis.ru/uploads/1283929429\_5076.pdf
4. Оптимальное сохранение контрольных точек на локальные устройства хранения URL: http://2015.russianscdays.org/files/pdf/288.pdf
5. Отказоустойчивая реализация метода молекулярной динамики на примере одного приложения URL: http://ceur-ws.org/Vol-1576/058.pdf
6. Building and using an Fault Tolerant MPI implementation URL: http://hpc.sagepub.com/content/18/3/353.abstract
7. A Proposal for User-Level Failure Mitigation in the MPI-3 Standard URL: http://icl.cs.utk.edu/news\_pub/submissions/mpi3ft.pdf
8. A Comprehensive User-level Checkpointing Strategy for MPI Applications URL: https://pdfs.semanticscholar.org/4ec7/0cf657913023001279af685e601a0f85dcea.pdf

ПРИЛОЖЕНИЕ

checkpoint\_lib.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* C - check \*/

/\* P - point \*/

/\* L - library \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "checkpoint\_lib.h"

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Global variables \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double cpl\_start\_time = 0.0;

void \*\*cpl\_checkpoint\_table;

int cpl\_size = 0;

int cpl\_time = 0;

int cpl\_counter = 0;

static struct itimerval cpl\_timer;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Timer \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline void timer\_init\_()

{

cpl\_timer.it\_interval.tv\_sec = cpl\_time; // interval

cpl\_timer.it\_interval.tv\_usec = 0;

cpl\_timer.it\_value.tv\_sec = cpl\_time; // time until next expiration

cpl\_timer.it\_value.tv\_usec = 0;

setitimer(ITIMER\_REAL, &cpl\_timer, NULL);

}

inline void timer\_stop\_()

{

cpl\_timer.it\_interval.tv\_sec = 0;

cpl\_timer.it\_value.tv\_sec = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Measure time \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline double wtime\_()

{

struct timeval t;

gettimeofday(&t, NULL);

return (double)t.tv\_sec + (double)t.tv\_usec \* 1E-6;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Aditional internal mpi functions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int get\_comm\_rank\_\_()

{

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

return rank;

}

static int get\_comm\_size\_\_()

{

int size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

return size;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Work with files \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void open\_snapshot\_file\_(MPI\_File \*snapshot, int phase)

{

char file\_name[256] = { 0 };

/\*

\* 1\_1\_1.3456 => [PHASE\_OF\_CALCULATION]\_[RANK]\_[CHECKPOINT\_TIME]

\*

\* PHASE\_OF\_CALCULATION - global state, need for 'goto'

\* RANK - process rank

\* CHECKPOINT\_TIME - each 'PHASE\_OF\_CALCULATION' could reach many times

\*/

sprintf(file\_name,"snapshot/%d\_%d\_%f", phase, get\_comm\_rank\_\_(), wtime\_() - cpl\_start\_time);

MPI\_File\_open( MPI\_COMM\_WORLD, file\_name,

MPI\_MODE\_CREATE|MPI\_MODE\_WRONLY,

MPI\_INFO\_NULL, snapshot );

}

void close\_snapshot\_file\_(MPI\_File \*snapshot)

{

MPI\_File\_close(snapshot);

}

void write\_to\_snapshot\_(MPI\_File file, void \*data, int n, MPI\_Datatype type)

{

MPI\_Status status;

MPI\_File\_write(file, data, n, type, &status);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Get last snapshot file \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int get\_lastcheckpoint\_rank\_\_(char \*file)

{

int j, i, checkpoint\_rank;

char tmp\_rank[10] = { 0 };

for (j = 0, i = 2; i < strlen(file); i++, j++) {

if (file[i] != '\_') {

tmp\_rank[j] = file[i];

} else {

break;

}

}

sscanf(tmp\_rank, "%d", &checkpoint\_rank);

return checkpoint\_rank;

}

static void get\_lastcheckpoint\_time\_\_(char \*a, char \*b)

{

int j, i;

double x, y;

char time\_last[10] = { 0 };

char time\_cur[10] = { 0 };

for (j = 0, i = 4; i < strlen(b); i++, j++) {

time\_last[j] = a[i];

time\_cur[j] = b[i];

}

sscanf(time\_last, "%lf", &x);

sscanf(time\_cur, "%lf", &y);

// Figure out which checkpoint is 'older'

if (x < y) {

strcpy(a, b);

}

}

int get\_last\_snapshot\_(char \*last\_checkpoint)

{

int myrank = get\_comm\_rank\_\_();

DIR \*dir;

struct dirent \*file;

dir = opendir("./snapshot"); // open current directory

if (dir) {

while (file = readdir(dir)) {

// Skip '.' '..' directory

if (strlen(file->d\_name) < 3) {

continue;

}

if (file->d\_name[1] != '\_') {

continue;

}

// Work only with myrank checkpoint

if (myrank == get\_lastcheckpoint\_rank\_\_(file->d\_name)) {

//printf("[DEBUG] Found for rank %d - %s\n", myrank, file->d\_name);

// Work only with greater checkpoint phase

if (last\_checkpoint[0] <= file->d\_name[0]) {

get\_lastcheckpoint\_time\_\_(last\_checkpoint, file->d\_name);

}

//printf("[DEBUG] %s\n", last\_checkpoint);

}

}

closedir(dir);

} else {

fprintf(stderr, "can't open current directory\n");

}

printf("Rankd %d, file %s, phase %d\n", myrank, last\_checkpoint, last\_checkpoint[0] - '0');

return last\_checkpoint[0] - '0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Get checkpoint index by checkpoint name \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_checkpoint\_idx\_by\_name\_(void \*\*table, int size, void \*name)

{

int i;

for (i = 0; i < size; i++) {

if (table[i] == name) {

break;

}

}

return i;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Init \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \*\*init\_table\_(int size)

{

void \*\*jump\_table = (void\*\*) malloc (sizeof(void\*) \* size);

if (!jump\_table) {

fprintf(stderr, "[ERROR] can't allocate memory for CPL\_GLOBAL\_JUMP\_TABLE\n");

exit(1);

}

mkdir("snapshot", 0777);

return jump\_table;

}

#ifndef \_CHECKPOINT\_LIB\_H\_

#define \_CHECKPOINT\_LIB\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* C - check \*/

/\* P - point \*/

/\* L - library \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <signal.h>

#include <sys/time.h>

#include <mpi.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Global variables \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern double cpl\_start\_time;

extern void \*\*cpl\_checkpoint\_table;

extern int cpl\_size;

extern int cpl\_time;

extern int cpl\_counter;

enum {

CPL\_CHECKPOINT\_MODE = 0,

CPL\_RECOVERY\_MODE = 1

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Initializing checkpoint library macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_INIT(size, time, func) \

signal(SIGALRM, func); \

cpl\_size = time; \

cpl\_counter = 0; \

cpl\_size = size; \

cpl\_start\_time = wtime\_(); \

cpl\_checkpoint\_table = init\_table\_(cpl\_size); \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Declaration checkpoint macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_DECLARATE\_CHECKPOINT(name) \

cpl\_checkpoint\_table[cpl\_counter++] = name; \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Control flow-macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_GO\_TO\_CHECKPOINT(idx) \

goto \*cpl\_checkpoint\_table[idx]; \

#define CPL\_SET\_CHECKPOINT(checkpoint\_name) \

checkpoint\_name : \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Checkpoint-save macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_FILE\_OPEN(file, phase) \

open\_snapshot\_file\_(file, phase); \

#define CPL\_FILE\_CLOSE(file) \

close\_snapshot\_file\_(file); \

#define CPL\_SAVE\_SNAPSHOT(file, data, n, type) \

write\_to\_snapshot\_(file, data, n, type); \

#define CPL\_GET\_SNAPSHOT(snapshot) \

get\_last\_snapshot\_(snapshot); \

#define CPL\_SAVE\_STATE(checkpoint, user\_save\_callback) \

user\_save\_callback(get\_checkpoint\_idx\_by\_name\_(cpl\_checkpoint\_table, cpl\_size, checkpoint));\

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Timer \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPL\_TIMER\_INIT() \

timer\_init\_(); \

#define CPL\_TIMER\_STOP() \

timer\_stop\_(); \

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Prototypes \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \*\*init\_table\_(int size);

double wtime\_();

void timer\_init\_();

void timer\_stop\_();

void write\_to\_snapshot\_(MPI\_File file, void \*data, int n, MPI\_Datatype type);

int get\_last\_snapshot\_(char \*last\_checkpoint);

int get\_checkpoint\_idx\_by\_name\_(void \*\*table, int size, void \*name);

void open\_snapshot\_file\_(MPI\_File \*snapshot, int phase);

void close\_snapshot\_file\_(MPI\_File \*snapshot);

#endif /\* \_CHECKPOINT\_LIB\_H\_ \*/

heat-2d.c

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#include <math.h>

#include <mpi.h>

#include <string.h>

#include "../checkpoint\_lib/checkpoint\_lib.h"

#define EPS 0.001

#define PI 3.14159265358979323846

#define NELEMS(x) (sizeof((x)) / sizeof((x)[0]))

#define IND(i, j) ((i) \* (nx + 2) + (j))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Global variables \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int is\_time\_to\_save = 0;

int options = CPL\_CHECKPOINT\_MODE;

//int options = CPL\_RECOVERY\_MODE;

int niters = 0;

int ny, nx;

double ttotal = 0.0;

double thalo = 0.0;

double treduce = 0.0;

double \*local\_grid = NULL;

double \*local\_newgrid = NULL;

void \*xcalloc(size\_t nmemb, size\_t size)

{

void \*p = calloc(nmemb, size);

if (p == NULL) {

fprintf(stderr, "No enough memory\n");

MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);

}

return p;

}

int get\_block\_size(int n, int rank, int nprocs)

{

int s = n / nprocs;

if (n % nprocs > rank)

s++;

return s;

}

int get\_sum\_of\_prev\_blocks(int n, int rank, int nprocs)

{

int rem = n % nprocs;

return n / nprocs \* rank + ((rank >= rem) ? rem : rank);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* User defiend callbacks \*/

/\* Additional functions helps save and get data \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline static void user\_save\_callback(int phase)

{

MPI\_File local\_snapshot;

CPL\_FILE\_OPEN(&local\_snapshot, phase);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &nx, 1, MPI\_INT);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &ny, 1, MPI\_INT);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, local\_grid, ((ny + 2) \* (nx + 2)), MPI\_DOUBLE);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &ttotal, 1, MPI\_DOUBLE);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &thalo, 1, MPI\_DOUBLE);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &treduce, 1, MPI\_DOUBLE);

CPL\_SAVE\_SNAPSHOT(local\_snapshot, &niters, 1, MPI\_INT);

CPL\_FILE\_CLOSE(&local\_snapshot);

is\_time\_to\_save = 0;

}

inline static int checkpoint\_get(double \*local\_grid,

int size,

double \*ttotal,

double \*thalo,

double \*treduce,

int \*niters)

{

int nx, ny;

char last\_chechkpoint[] = { "0\_0\_0.000000" };

int phase = CPL\_GET\_SNAPSHOT(last\_chechkpoint);

char last\_chechkpoint\_path[256] = { "snapshot/" };

strcat(last\_chechkpoint\_path, last\_chechkpoint);

FILE \* file = fopen(last\_chechkpoint\_path, "rb");

if (!file) {

fprintf(stderr, "Can't read snapshot\n");

exit(1);

}

// copy the file into the buffer:

fread(&nx, sizeof(int), 1, file);

fread(&ny, sizeof(int), 1, file);

if (size != ((ny + 2) \* (nx + 2))) {

fprintf(stderr, "Snapshot size not match\n");

fclose(file);

exit(1);

} else {

fprintf(stderr, "Snapshot size match\n");

}

fread(local\_grid, sizeof(double), size, file);

fread(ttotal, sizeof(double), 1, file);

fread(thalo, sizeof(double), 1, file);

fread(treduce, sizeof(double), 1, file);

fread(niters, sizeof(int), 1, file);

fclose(file);

return phase;

}

void time\_handler(int sig)

{

is\_time\_to\_save = 1;

}

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

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Initialize checkpoint library \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int checkpoint\_numbers = 2;

int timers\_time = 5;

CPL\_INIT(checkpoint\_numbers, timers\_time, time\_handler);

CPL\_DECLARATE\_CHECKPOINT(&&phase\_one);

CPL\_DECLARATE\_CHECKPOINT(&&phase\_two);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Local variables \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char procname[MPI\_MAX\_PROCESSOR\_NAME];

int commsize, rank, rankx, ranky, rows, cols, px, py, namelen;

int coords[2];

int dims[2] = { 0, 0 };

int periodic[2] = { 0, 0 };

int left, right, top, bottom;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Initialize \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Initialize args, rank, commsize \*/

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &commsize);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

/\* Get processor name \*/

MPI\_Get\_processor\_name(procname, &namelen);

/\* Create 2D grid of processes: commsize = px \* py \*/

MPI\_Comm cartcomm;

MPI\_Dims\_create(commsize, 2, dims);

px = dims[0];

py = dims[1];

/\* Make start time point \*/

ttotal = MPI\_Wtime();

if (px < 2 || py < 2) {

fprintf(stderr, "Invalid number of processes %d: px %d and py %d must be greater than 1\n", commsize, px, py);

MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);

}

MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periodic, 0, &cartcomm);

MPI\_Cart\_coords(cartcomm, rank, 2, coords);

rankx = coords[0];

ranky = coords[1];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Broadcast command line arguments \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (rank == 0) {

rows = (argc > 1) ? atoi(argv[1]) : py \* 100;

cols = (argc > 2) ? atoi(argv[2]) : px \* 100;

if (rows < py) {

fprintf(stderr, "Number of rows %d less then number of py processes %d\n", rows, py);

MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);

}

if (cols < px) {

fprintf(stderr, "Number of cols %d less then number of px processes %d\n", cols, px);

MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);

}

int args[2] = { rows, cols };

MPI\_Bcast(&args, NELEMS(args), MPI\_INT, 0, MPI\_COMM\_WORLD);

} else {

int args[2];

MPI\_Bcast(&args, NELEMS(args), MPI\_INT, 0, MPI\_COMM\_WORLD);

rows = args[0];

cols = args[1];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Allocate memory for local 2D subgrids with halo cells \*/

/\* [0..ny + 1][0..nx + 1] \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ny = get\_block\_size(rows, ranky, py);

nx = get\_block\_size(cols, rankx, px);

local\_grid = xcalloc((ny + 2) \* (nx + 2), sizeof(\*local\_grid));

local\_newgrid = xcalloc((ny + 2) \* (nx + 2), sizeof(\*local\_newgrid));

// Fill boundary points:

// - left and right borders are zero filled

// - top border: u(x, 0) = sin(pi \* x)

// - bottom border: u(x, 1) = sin(pi \* x) \* exp(-pi)

double dx = 1.0 / (cols - 1.0);

int sj = get\_sum\_of\_prev\_blocks(cols, rankx, px);

if (ranky == 0) {

// Initialize top border: u(x, 0) = sin(pi \* x)

for (int j = 1; j <= nx; j++) {

// Translate col index to x coord in [0, 1]

double x = dx \* (sj + j - 1);

int ind = IND(0, j);

local\_newgrid[ind] = local\_grid[ind] = sin(PI \* x);

}

}

if (ranky == (py - 1)) {

// Initialize bottom border: u(x, 1) = sin(pi \* x) \* exp(-pi)

for (int j = 1; j <= nx; j++) {

// Translate col index to x coord in [0, 1]

double x = dx \* (sj + j - 1);

int ind = IND(ny + 1, j);

local\_newgrid[ind] = local\_grid[ind] = sin(PI \* x) \* exp(-PI);

}

}

// Neighbours

MPI\_Cart\_shift(cartcomm, 0, 1, &left, &right);

MPI\_Cart\_shift(cartcomm, 1, 1, &top, &bottom);

// Left and right borders type

MPI\_Datatype col;

MPI\_Type\_vector(ny, 1, nx + 2, MPI\_DOUBLE, &col);

MPI\_Type\_commit(&col);

// Top and bottom borders type

MPI\_Datatype row;

MPI\_Type\_contiguous(nx, MPI\_DOUBLE, &row);

MPI\_Type\_commit(&row);

MPI\_Request reqs[8];

if (options == CPL\_CHECKPOINT\_MODE) {

CPL\_SAVE\_STATE(&&phase\_one, user\_save\_callback);

} else if (options == CPL\_RECOVERY\_MODE) {

int phase = checkpoint\_get(local\_grid, ((ny + 2) \* (nx + 2)), &ttotal, &thalo, &treduce, &niters);

// Jumping to checkpoint

CPL\_GO\_TO\_CHECKPOINT(phase);

}

CPL\_SET\_CHECKPOINT(phase\_one);

CPL\_TIMER\_INIT();

is\_time\_to\_save = 1;

for (;;) {

niters++;

// Update interior points

for (int i = 1; i <= ny; i++) {

for (int j = 1; j <= nx; j++) {

local\_newgrid[IND(i, j)] =

(local\_grid[IND(i - 1, j)] + local\_grid[IND(i + 1, j)] +

local\_grid[IND(i, j - 1)] + local\_grid[IND(i, j + 1)]) \* 0.25;

}

}

// Check termination condition

double maxdiff = 0;

for (int i = 1; i <= ny; i++) {

for (int j = 1; j <= nx; j++) {

int ind = IND(i, j);

maxdiff = fmax(maxdiff, fabs(local\_grid[ind] - local\_newgrid[ind]));

}

}

// Swap grids (after termination local\_grid will contain result)

double \*p = local\_grid;

local\_grid = local\_newgrid;

local\_newgrid = p;

treduce -= MPI\_Wtime();

MPI\_Allreduce(MPI\_IN\_PLACE, &maxdiff, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);

treduce += MPI\_Wtime();

if (maxdiff < EPS) {

break;

}

// Halo exchange: T = 4 \* (a + b \* (rows / py)) + 4 \* (a + b \* (cols / px))

thalo -= MPI\_Wtime();

MPI\_Irecv(&local\_grid[IND(0, 1)], 1, row, top, 0, cartcomm, &reqs[0]); // top

MPI\_Irecv(&local\_grid[IND(ny + 1, 1)], 1, row, bottom, 0, cartcomm, &reqs[1]); // bottom

MPI\_Irecv(&local\_grid[IND(1, 0)], 1, col, left, 0, cartcomm, &reqs[2]); // left

MPI\_Irecv(&local\_grid[IND(1, nx + 1)], 1, col, right, 0, cartcomm, &reqs[3]); // right

MPI\_Isend(&local\_grid[IND(1, 1)], 1, row, top, 0, cartcomm, &reqs[4]); // top

MPI\_Isend(&local\_grid[IND(ny, 1)], 1, row, bottom, 0, cartcomm, &reqs[5]); // bottom

MPI\_Isend(&local\_grid[IND(1, 1)], 1, col, left, 0, cartcomm, &reqs[6]); // left

MPI\_Isend(&local\_grid[IND(1, nx)], 1, col, right, 0, cartcomm, &reqs[7]); // right

MPI\_Waitall(8, reqs, MPI\_STATUS\_IGNORE);

thalo += MPI\_Wtime();

if (is\_time\_to\_save)

CPL\_SAVE\_STATE(&&phase\_one, user\_save\_callback);

MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);

}

CPL\_SAVE\_STATE(&&phase\_two, user\_save\_callback);

CPL\_SET\_CHECKPOINT(phase\_two);

MPI\_Type\_free(&row);

MPI\_Type\_free(&col);

free(local\_newgrid);

free(local\_grid);

ttotal += MPI\_Wtime();

if (rank == 0) {

printf("# Heat 2D (mpi): grid: rows %d, cols %d, procs %d (px %d, py %d)\n", rows, cols, commsize, px, py);

}

printf("# P %4d (%2d, %2d) on %s: grid ny %d nx %d, total %.6f,"

" mpi %.6f (%.2f) = allred %.6f (%.2f) + halo %.6f (%.2f)\n",

rank, rankx, ranky, procname, ny, nx, ttotal, treduce + thalo,

(treduce + thalo) / ttotal, treduce, treduce / (treduce + thalo),

thalo, thalo / (treduce + thalo));

double prof[3] = { ttotal, treduce, thalo };

if (rank == 0) {

MPI\_Reduce(MPI\_IN\_PLACE, prof, NELEMS(prof), MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

printf("# procs %d : grid %d %d : niters %d : total time %.6f :"

" mpi time %.6f : allred %.6f : halo %.6f\n",

commsize, rows, cols, niters, prof[0], prof[1] + prof[2], prof[1], prof[2]);

} else {

MPI\_Reduce(prof, NULL, NELEMS(prof), MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}