Федеральное агенство связи

Федеральное государственное бюджетное образовательное учереждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

по дисциплине “Параллельное программирование”

на тему

Разработка библиотеки отказоустойчивости для вычислительных систем с массовым параллелизмом на примере одного приложения

Выполнил студент Марков В.А.
Ф.И.О.

Группы МГ-165

Работу принял к.т.н. М.Г. Курносов
подпись

Защищена Оценка

Новосибирск – 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для существующих вычислительных систем петафлопсного уровня производительности и проектируемых систем эксафлопсного 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

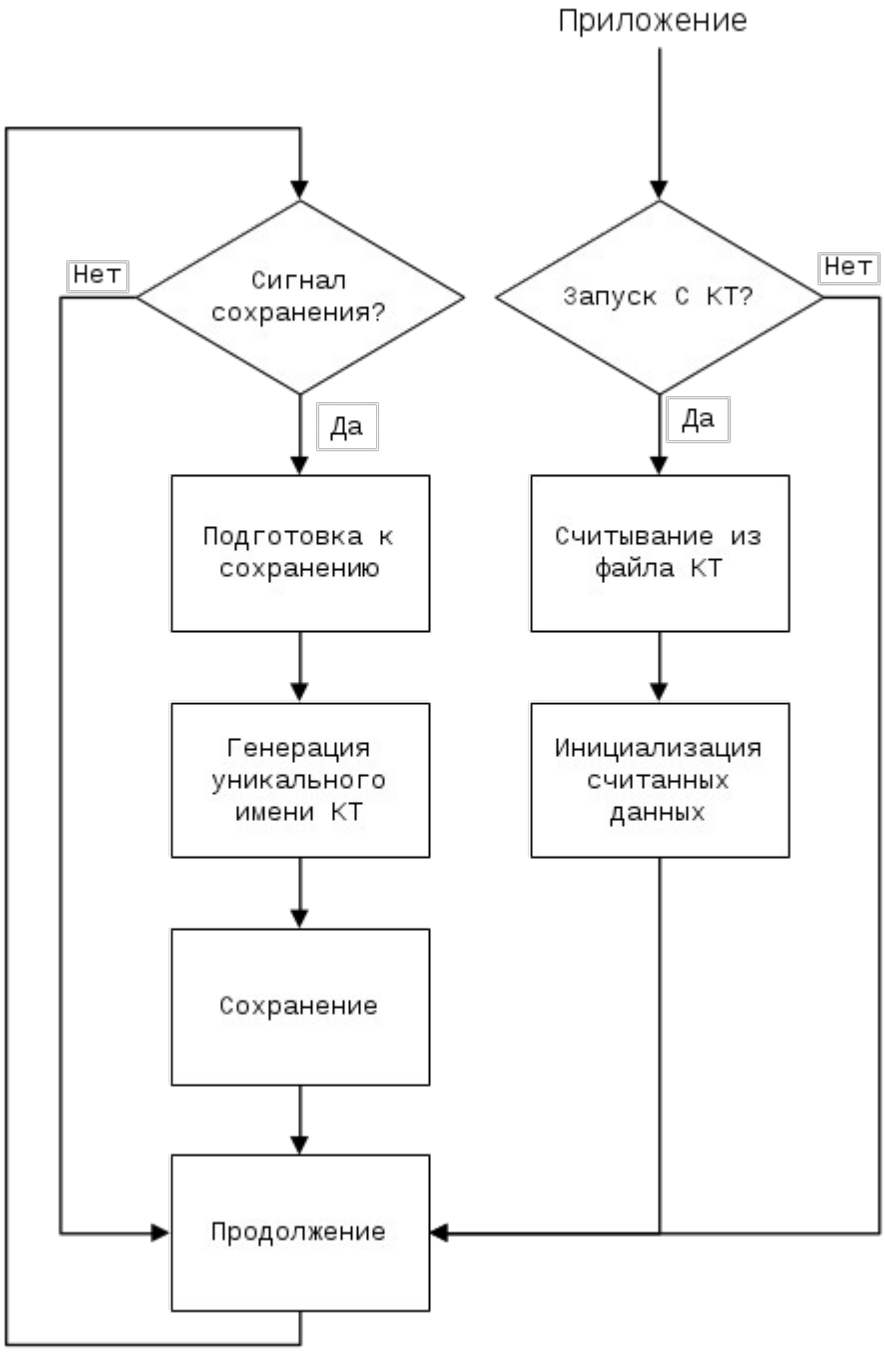

Рисунок 1 – Алгоритм работы КТ

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 на примере приложения, расчет теплопроводности двумерной пластины.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек. А. А. Бондаренко, М. В. Якобовский, 2015 г. 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. . А. Бондаренко, М. В. Якобовский, 2015 г. 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. А.Ю. Поляков, А.А. Данекина, 2010 г. //URL: http://vestnik.sibsutis.ru/uploads/1283929429_5076.pdf
4. Оптимальное сохранение контрольных точек на локальные устройства хранения. А. А. Бондаренко, М.В. Якобовский, 2015 г. //URL: http://2015.russianscdays.org/files/pdf/288.pdf
5. Отказоустойчивая реализация метода молекулярной динамики на примере одного приложения. А.А. Бондаренко, В.О. Подрыга, С.В. Поляков, М.В. Якобовский, 2016 г. //URL: http://ceur-ws.org/Vol-1576/058.pdf
6. Building and using an Fault Tolerant MPI implementation. Graham E. Fagg Jack J. Dongarra, 2004 г. //URL: http://hpc.sagepub.com/content/18/3/353.abstract
7. A Proposal for User-Level Failure Mitigation in the MPI-3 Standard. Wesley Bland, George Bosilca, 2012 г. //URL: http://icl.cs.utk.edu/news_pub/submissions/mpi3ft.pdf
8. A Comprehensive User-level Checkpointing Strategy for MPI Applications. John Paul Walters, 2007 г. //URL: https://pdfs.semanticscholar.org/4ec7/0cf657913023001279af685e601a0f85dcea.pdf