Наиболее часто встречаемые ошибки в OpenMP-программах. Функциональная отладка OpenMP-программ

Параллельное программирование с OpenMP

Бахтин Владимир Александрович
Доцент кафедры системного программированния факультета ВМК, МГУ им. М. В. Ломоносова
К.ф.-м.н., зав. сектором Института прикладной математики им М.В.Келдыша РАН

Содержание

- Трудно обнаруживаемые ошибки типа Data Race (конфликт доступа к данным) и Race Condition.
- Ошибки типа Deadlock (взаимная блокировка нитей).
- Ошибки, связанные с использованием неинициализированных переменных.
- Автоматизированный поиск ошибок в OpenMP-программах при помощи Intel Thread Checker (Intel Parallel Inspector), Sun Studio Thread Analyzer (Oracle Solaris Studio) и Valgrind.

Конфликт доступа к данным

При взаимодействии через общую память нити должны синхронизовать свое выполнение.

```
#pragma omp parallel
{
        sum = sum + val;
}
```

Время	Thread 0	Thread 1
1	LOAD R1,sum	
2	LOAD R2,val	
3	ADD R1,R2	LOAD R3,sum
4	STORE R1,sum	LOAD R4,val
5		ADD R3,R4
6		STORE R3,sum

Результат зависит от порядка выполнения команд. Требуется взаимное исключение критических интервалов.

Конфликт доступа к данным

Ошибка возникает при одновременном выполнении следующих условий:

- **Две или более нитей обращаются к одной и той же ячейке памяти.**
- **По крайней мере, один из этих доступов к памяти является записью.**
- **Нити не синхронизируют свой доступ к данной ячейки памяти.**

При одновременном выполнении всех трех условий порядок доступа становится неопределенным.

Конфликт доступа к данным

Использование различных компиляторов (различных опций оптимизации, включение/отключение режима отладки при компиляции программы), применение различных стратегий планирования выполнения нитей в различных ОС, может приводить к тому, что в каких-то условиях (например, на одной вычислительной машине) ошибка не будет проявляться, а в других (на другой машине) – приводить к некорректной работе программы.

От запуска к запуску программа может выдавать различные результаты в зависимости от порядка доступа.

Отловить такую ошибку очень тяжело.

Причиной таких ошибок, как правило являются:

- **неверное определение класса переменной,**
- **отсутствие синхронизации.**

Конфликт доступа к данным

```
#define N
float a[N], tmp;
#pragma omp parallel
{
    #pragma omp for
    for(int i=0; i<N;i++) {
        tmp= a[i]*a[i];
        a[i]=1-tmp;
    }
}
```

Конфликт доступа к данным

```
#define N
float a[N], tmp;
#pragma omp parallel
{
    #pragma omp for
    for(int i=0; i<N;i++) {
        tmp= a[i]*a[i];
        a[i]=1-tmp;
    }
}
```

```
#define N
float a[N], tmp;
#pragma omp parallel
{
 #pragma omp for private(tmp)
    for(int i=0; i<N;i++) {
        tmp= a[i]*a[i];
        a[i]=1-tmp;
    }
}
```

Конфликт доступа к данным

file1.c

```
int counter = 0;
#pragma omp threadprivate(counter)

int increment_counter()
{
counter++;
return(counter);
}
```

file2.c

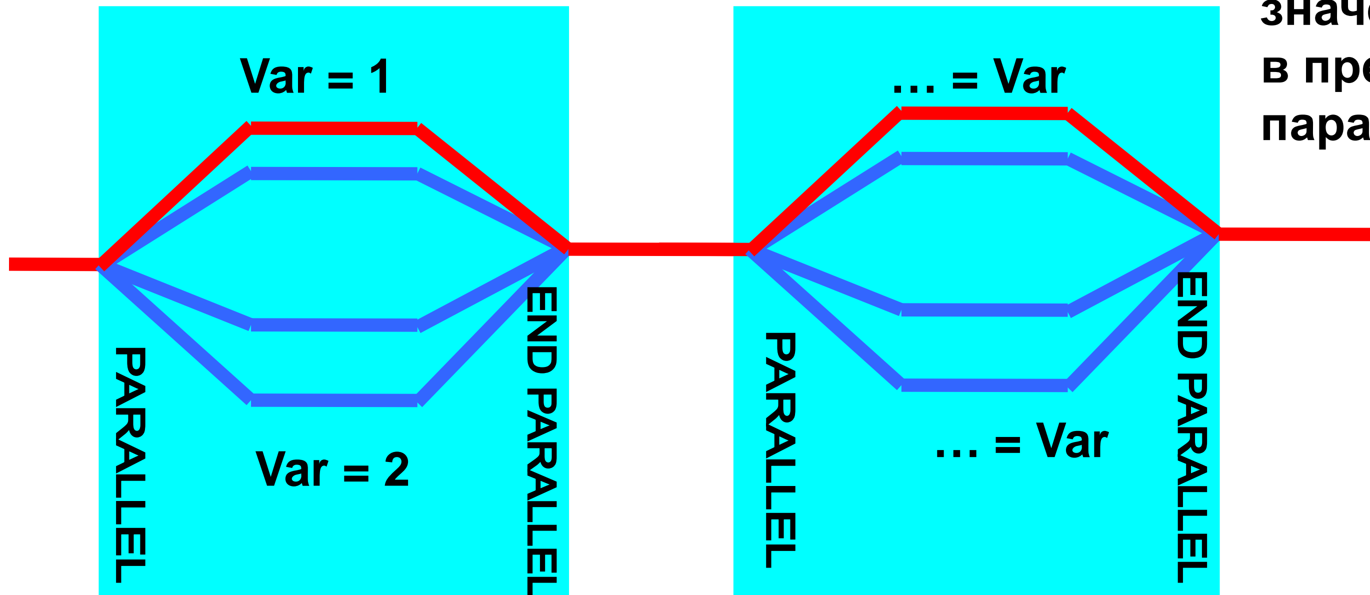
```
extern int counter;

int decrement_counter()
{
counter--;
return(counter);
}
```

Директива threadprivate

threadprivate – переменные сохраняют глобальную область видимости внутри каждой нити

#pragma omp threadprivate (Var)

Если количество нитей не изменилось, то каждая нить получит значение, посчитанное в предыдущей параллельной области.



Конфликт доступа к данным

file1.c

```
int counter = 0;
#pragma omp threadprivate(counter)

int increment_counter()
{
counter++;
return(counter);
}
```

file2.c

```
extern int counter;
#pragma omp threadprivate(counter)
int decrement_counter()
{
counter--;
return(counter);
}
```

Конфликт доступа к данным

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], B[N], maxval;
#pragma omp parallel
{
    #pragma omp master
         maxval = 0.0;
    #pragma omp for
     for(int i=1; i<N-1;i++) {
        B[i] = (A[i-1] + A[i+1]) / 2.;
        maxval =  Max(A[i]-B[i],maxval);
     }
}
```

Конфликт доступа к данным

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], B[N], maxval;
#pragma omp parallel
{
    #pragma omp master
        maxval = 0.0;
    #pragma omp for
     for(int i=1; i<N-1;i++) {
        B[i] = (A[i-1] + A[i+1]) / 2;
        maxval =  Max(A[i]-B[i],maxval);
     }
}
```

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], B[N], maxval;
#pragma omp parallel
{
    #pragma omp master
        maxval = 0.0;
    #pragma omp for
     for(int i=1; i<N-1;i++) {
        B[i] = (A[i-1] + A[i+1]) / 2;
        #pragma omp critical
          maxval =  Max(A[i] -B[i],maxval);
     }
}
```

Конфликт доступа к данным

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], B[N], maxval;
#pragma omp parallel
{
    #pragma omp master
         maxval = 0.0;
    #pragma omp for
     for(int i=1; i<N-1;i++) {
        B[i] = (A[i-1] + A[i+1]) / 2;
        maxval =  Max(A[i]-B[i],maxval);
     }
}
```

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], B[N], maxval;
#pragma omp parallel
{
    #pragma omp master
         maxval = 0.0;
    #pragma omp barrier
    #pragma omp for
     for(int i=1; i<N-1;i++) {
         B[i] = (A[i-1] + A[i+1]) / 2;
         #pragma omp critical
             maxval =  Max(A[i]-B[i],maxval);
     }
}
```

Конфликт доступа к данным

```
void example(int n, int m, float *a, float *b, float *c, float *z)
{
    int i;
    float sum = 0.0;
    #pragma omp parallel
    {
        #pragma omp for schedule(runtime) nowait
        for (i=0; i<m; i++) {
            c[i] = (a[i] + b[i]) / 2.0;
        }
        #pragma omp for schedule(runtime) nowait
          for (i=0; i<n; i++)
             z[i] = sqrt(c[i]);
    }
}
```

Конфликт доступа к данным

```
void example(int n, int m, float *a, float *b, float *c, float *z)
{
    int i;
    float sum = 0.0;
    #pragma omp parallel
    {
        #pragma omp for schedule(runtime)
        for (i=0; i<m; i++) {
            c[i] = (a[i] + b[i]) / 2.0;
        }
        #pragma omp for schedule(runtime) nowait
          for (i=0; i<n; i++)
             z[i] = sqrt(c[i]);
    }
}
```

Конфликт доступа к данным

```
void example(int n, float *a, float *b, float *c, float *z)
{
   int i;
   float sum = 0.0;
   #pragma omp parallel
   {
      #pragma omp for nowait reduction (+: sum)
      for (i=0; i<n; i++) {
         c[i] = (a[i] + b[i]) / 2.0;
         sum += c[i];
      }
      #pragma omp for nowait
        for (i=0; i<n; i++)
          z[i] = sqrt(b[i]);
       #pragma omp master
         printf ("Sum of array C=%g\n",sum);

   }
}
```

Конфликт доступа к данным

```
void example(int n, float *a, float *b, float *c, float *z)
{
    int i;
    float sum = 0.0;
    #pragma omp parallel
    {
        #pragma omp for schedule(static) nowait reduction (+: sum)
        for (i=0; i<n; i++) {
            c[i] = (a[i] + b[i]) / 2.0;
            sum += c[i];
        }
        #pragma omp for schedule(static) nowait
          for (i=0; i<n; i++)
             z[i] = sqrt(b[i]);
         #pragma omp barrier
         #pragma omp master
             printf (“Sum of array C=%g\n”,sum);
    }
}
```

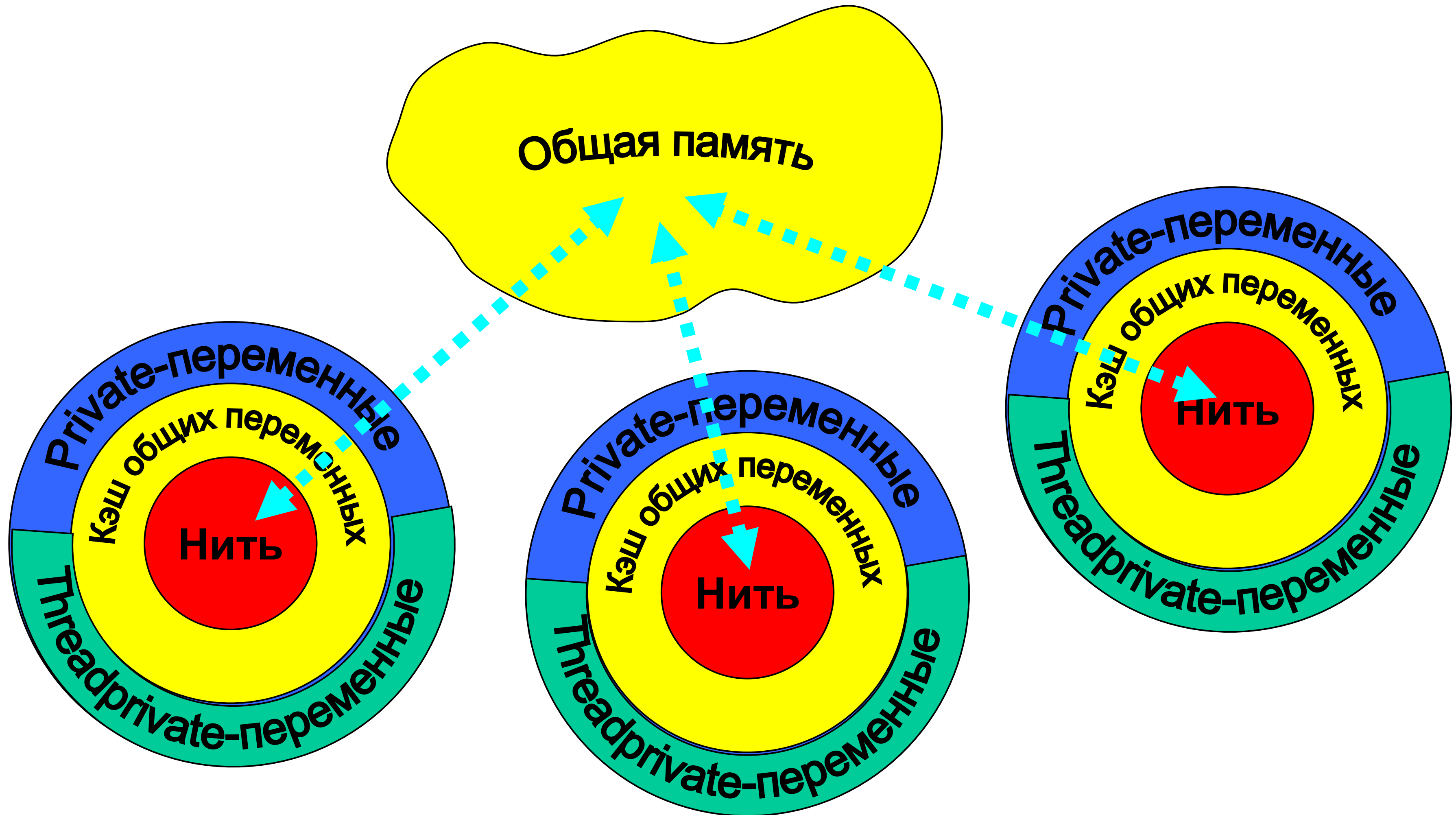
Распределение циклов с зависимостью по данным. Организация конвейерного выполнения цикла.

```
for(int i = 1; i < N; i++)
  for(int j = 1; j < N; j++)
    a[i][j] = (a[i-1][j] + a[i][j-1] + a[i+1][j] + a[i][j+1]) / 4
```

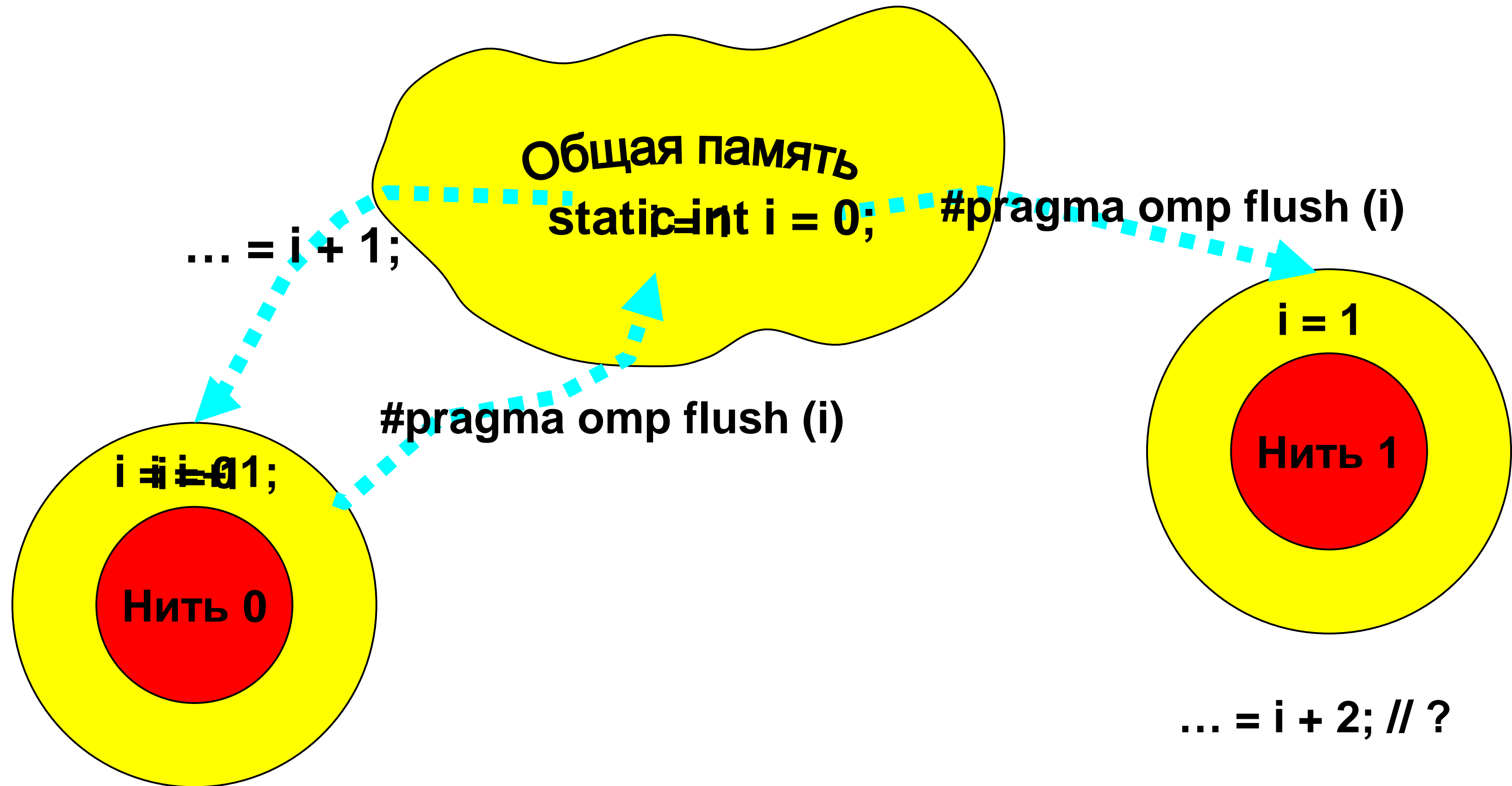
Распределение циклов с зависимостью по данным. Организация конвейерного выполнения цикла.

```
  int isync[NUMBER_OF_THREADS];
  int iam, numt, limit;
#pragma omp parallel private(iam,numt,limit)
  {
  iam = omp_get_thread_num ();
  numt = omp_get_num_threads ();
  limit=min(numt-1,N-2);
  isync[iam]=0;
#pragma omp barrier
  for (int i=1; i<N; i++) {
    if ((iam>0) && (iam<=limit)) {
       for (;isync[iam-1]==0;) ;
       isync[iam-1]=0;
    }
#pragma omp for schedule(static) nowait
    for (int j=1; j<N; j++) {
      a[i][j]=(a[i-1][j] + a[i][j-1] + a[i+1][j] +
                 a[i][j+1])/4;
    }
    if (iam<limit) {
       for (;isync[iam]==1;);
       isync[iam]=1;
    }
  }
}
```

Модель памяти в OpenMP



Модель памяти в OpenMP



Консистентность памяти в OpenMP

Корректная последовательность работы нитей с переменной:

Нить0 записывает значение переменной - write(var)

Нить0 выполняет операцию синхронизации – flush (var)

Нить1 выполняет операцию синхронизации – flush (var)

Нить1 читает значение переменной – read (var)

Директива flush:

#pragma omp flush *[(список переменных)]* - для Си

По умолчанию все переменные приводятся в консистентное состояние **(#pragma omp flush)**:

- При барьерной синхронизации.
- При входе и выходе из конструкций **parallel**, **critical** и **ordered**.
- При выходе из конструкций распределения работ (**for**, **single**, **sections**, **workshare**), если не указана клауза **nowait**.
- При вызове **omp_set_lock** и **omp_unset_lock**.
- При вызове **omp_test_lock**, **omp_set_nest_lock**, **omp_unset_nest_lock** и **omp_test_nest_lock**, если изменилось состояние семафора.

Распределение циклов с зависимостью по данным. Организация конвейерного выполнения цикла.

```
  int isync[NUMBER_OF_THREADS];
  int iam, numt, limit;
#pragma omp parallel private(iam,numt,limit)
  {
  iam = omp_get_thread_num ();
  numt = omp_get_num_threads ();
  limit=min(numt-1,N-2);
  isync[iam]=0;
#pragma omp barrier
  for (int i=1; i<N; i++) {
    if ((iam>0) && (iam<=limit)) {
       for (;isync[iam-1]==0;) {
          #pragma omp flush (isync)
       }
       isync[iam-1]=0;
       #pragma omp flush (isync)
    }
#pragma omp for schedule(static) nowait
    for (int j=1; j<N; j++) {
      a[i][j]=(a[i-1][j] + a[i][j-1] + a[i+1][j] +
               a[i][j+1])/4;
    }
    if (iam<limit) {
       for (;isync[iam]==1;) {
         #pragma omp flush (isync)
       }
       isync[iam]=1;
       #pragma omp flush (isync)
    }
  }
}
```

Конфликт доступа к данным

```
#define  ITMAX 20
#define  Max(a,b) ((a)>(b)?(a):(b))
double MAXEPS   = 0.5;
double grid[L][L], tmp[L][L],eps;
#pragma omp parallel
{
   for (int it=0;it<ITMAX; it++) {
      eps= 0.;
      #pragma omp for
      for (int i=1; i<N-1; i++ )
         for (int j=1; j<N-1; j++ ) {
            #pragma omp critical
               eps =  Max(fabs(tmp[i][j]-grid[i][j]),eps);
            grid[i][j] = tmp[i][j];
      }
      #pragma omp for
       for (int i=1; i<N-1; i++ )
          for (int j=1; j<N-1; j++ )
             tmp[i][j] = 0.25 * ( grid[i-1][j] + grid[i+1][j] + grid[i][j-1] + grid[i][j+1]);
       if (eps < MAXEPS) break;
   }
}
```

Конфликт доступа к данным

```
#define  ITMAX 20
#define  Max(a,b) ((a)>(b)?(a):(b))
double MAXEPS   = 0.5;
double grid[L][L], tmp[L][L],eps;
#pragma omp parallel
{
   for (int it=0;it<ITMAX; it++) {
      #pragma omp single
         eps= 0.;
      #pragma omp for
      for (int i=1; i<N-1; i++ )
         for (int j=1; j<N-1; j++ ) {
            #pragma omp critical
               eps =  Max(fabs(tmp[i][j]-grid[i][j]),eps);
            grid[i][j] = tmp[i][j];
      }
      #pragma omp for
       for (int i=1; i<N-1; i++ )
          for (int j=1; j<N-1; j++ )
             tmp[i][j] = 0.25 * ( grid[i-1][j] + grid[i+1][j] + grid[i][j-1] + grid[i][j+1]);
       if (eps < MAXEPS) break;
   }
}
```

Конфликт доступа к данным

```
#define  Max(a,b) ((a)>(b)?(a):(b))
double MAXEPS   = 0.5;
double grid[L][L], tmp[L][L],eps;
#pragma omp parallel
{
   for (int it=0;it<ITMAX; it++) {
     #pragma omp barrier
     #pragma omp single
        eps= 0.;
     #pragma omp for
     for (int i=1; i<N-1; i++ )
        for (int j=1; j<N-1; j++ ) {
           #pragma omp critical
              eps =  Max(fabs(tmp[i][j]-grid[i][j]),eps);
           grid[i][j] = tmp[i][j];
     }
     #pragma omp for
      for (int i=1; i<N-1; i++ )
         for (int j=1; j<N-1; j++ )
            tmp[i][j] = 0.25 * ( grid[i-1][j] + grid[i+1][j] + grid[i][j-1] + grid[i][j+1]);
      if (eps < MAXEPS) break;
   }
}
```

Взаимная блокировка нитей

```
#define N 10
int A[N],B[N], sum;
#pragma omp parallel num_threads(10)
{
    int iam=omp_get_thread_num();
    if (iam ==0) {
      #pragma omp critical (update_a)
        #pragma omp critical (update_b)
          sum +=A[iam];
    } else {
      #pragma omp critical (update_b)
        #pragma omp critical (update_a)
          sum +=B[iam];
    }
}
```

Семафоры в OpenMP

```
#include <omp.h>
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
int main ()
{
    omp_lock_t lck;
    float A[N], maxval;
    #pragma omp parallel
    {
        #pragma omp master
            maxval = 0.0;
        #pragma omp barrier
        #pragma omp for
        for(int i=0; i<N;i++) {
           omp_set_lock(&lck);
              maxval =  Max(A[i],maxval);
           omp_unset_lock(&lck);
        }
     }
     return 0;
}
```

Семафоры в OpenMP

```
#include <omp.h>
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
int main ()
{
    omp_lock_t lck;
    float A[N], maxval;
     omp_init_lock(&lck);
    #pragma omp parallel
    {
        #pragma omp master
            maxval = 0.0;
        #pragma omp barrier
        #pragma omp for
        for(int i=0; i<N;i++) {
           omp_set_lock(&lck);
              maxval =  Max(A[i],maxval);
           omp_unset_lock(&lck);
        }
     }
    omp_destroy_lock(&lck);
    return 0;
}
```

Взаимная блокировка нитей

```
#pragma omp parallel
{
    int iam=omp_get_thread_num();
    if (iam ==0) {
        omp_set_lock (&lcka);
        omp_set_lock (&lckb);
        x = x + 1;
        omp_unset_lock (&lckb);
        omp_unset_lock (&lcka);
     } else {
        omp_set_lock (&lckb);
        omp_set_lock (&lcka);
         x = x + 2;
         omp_unset_lock (&lcka);
         omp_unset_lock (&lckb);
    }
}
```

Взаимная блокировка нитей

```
#pragma omp parallel
{
    int iam=omp_get_thread_num();
    if (iam ==0) {
        omp_set_lock (&lcka);
        while (x<0); /*цикл ожидания*/
        omp_unset_lock (&lcka);
     } else {
        omp_set_lock (&lcka);
        x++;
        omp_unset_lock (&lcka);
     }
   }
}
```

Неинициализированные переменные

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], maxval, localmaxval;
maxval = localmaxval = 0.0;
#pragma omp parallel private (localmaxval)
{
    #pragma omp for
     for(int i=0; i<N;i++) {
          localmaxval =  Max(A[i],localmaxval);
     }
     #pragma omp critical
          maxval =  Max(localmaxval,maxval);
}
```

Неинициализированные переменные

```
#define N 100
#define  Max(a,b) ((a)>(b)?(a):(b))
float A[N], maxval, localmaxval;
maxval = localmaxval = 0.0;
#pragma omp parallel firstprivate (localmaxval)
{
    #pragma omp for
     for(int i=0; i<N;i++) {
          localmaxval =  Max(A[i],localmaxval);
     }
     #pragma omp critical
          maxval =  Max(localmaxval,maxval);
}
```

Неинициализированные переменные

```
int tmp = 0;
#pragma omp parallel
{
#pragma omp for firstprivate(tmp), lastprivate (tmp)
      for (int j = 0; j < 100; ++j) {
            if (j<98) tmp = j;
      }
printf(“%d\n”, tmp);
}
```

Неинициализированные переменные

```
static int counter;
#pragma omp threadprivate(counter)

int main () {
      counter = 0;
     #pragma omp parallel
      {
         counter++;
      }
}
```

Неинициализированные переменные

```
static int counter;
#pragma omp threadprivate(counter)

int main () {
      counter = 0;
      #pragma omp parallel copyin  (counter)
      {
         counter++;
      }
}
```

Автоматизированный поиск ошибок. Intel Thread Checker (Intel Parallel Inspector)

KAI Assure for Threads (Kuck and Associates)

Анализ программы основан на процедуре инструментации.

Инструментация **– вставка обращений для записи действий, потенциально способных привести к ошибкам: работа с памятью, вызовы операций синхронизации и работа с потоками.**

Может выполняться:

- **автоматически (бинарная инструментация) на уровне исполняемого модуля (а также dll-библиотеки)**
- **и/или по указанию программиста на уровне исходного кода (компиляторная инструментация Windows).**

Windows: /DEBUG /Zi

Linux: -g

Использовать динамические runtime-библиотеки (Windows: /MDd)

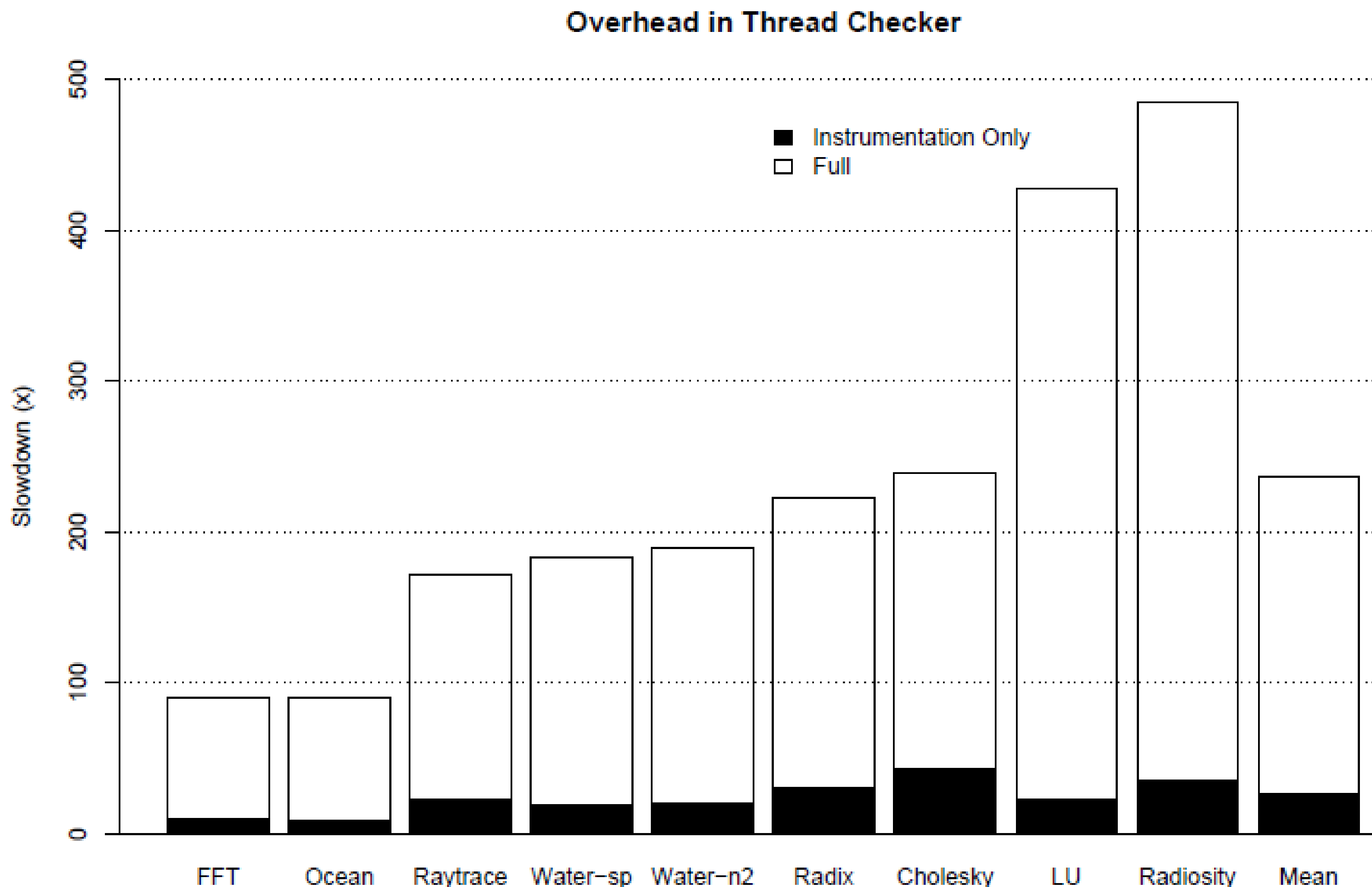
Автоматизированный поиск ошибок. Intel Thread Checker

Для каждой использованной в программе переменной сохраняется:

- **адрес переменной;**
- **тип использования (read или write);**
- **наличие/отсутствие операции синхронизации;**
- **номер строки и имя файла;**
- **call stack.**

Инструментация программы + большой объем сохраняемой информации для каждого обращения = существенные накладные расходы и замедление выполнения программы.

Пакет тестов SPLASH-2 (Stanford Parallel Applications for Shared Memory) на 4-х ядерной машине



http://iacoma.cs.uiuc.edu/iacoma-papers/asid06.pdf

Автоматизированный поиск ошибок. Sun Thread Analyzer

Инструментация программы:

cc -xinstrument=datarace -g -xopenmp=noopt test.c

Накопление информации о программе:

export OMP_NUM_THREADS=2

collect -r race ./a.out

collect -r deadlock ./a.out

collect -r all ./a.out

Получение результатов анализа программы

tha test.1.er => GUI

er_print test.1.er => интерфейс командной строки

Автоматизированный поиск ошибок. Sun Thread Analyzer

```
if (iam==0) {
        user_lock ();
        data = …
         …
} else {
        user_lock ();
        … = data;
        …
}
```

```
if (iam==0) {
        ptr1 = mymalloc(sizeof(data_t));
        ptr1->data = ...
        ...
        myfree(ptr1);
} else {
        ptr2 = mymalloc(sizeof(data_t));
        ptr2->data = ...
        ...
        myfree(ptr2);
}
```

Может выдавать сообщения об ошибках там где их нет

Intel Thread Checker и Sun Thread Analyzer

Программа	Jacobian Solver		Sparse Matrix-Vector multiplication		Adaptive Integration Solver	
	Mem	MFLOP/s	Mem	MFLOP/s	Mem	Time
Intel on 2 Threads	5	621	40	929	4	5.0 s
Intel Thread Checker on 2 Threads	115	0.9	1832	3.5	30	9.5 s
Intel Thread Checker –tcheck	115	3.1	-	-	-	-
Sun on 2 Threads	5	600	50	550	2	8.4 s
Sun Thread Analyzer on 2 Threads	125	1.1	2020	0.8	17	8.5 s

http://www.fz-juelich.de/nic-series/volume38/terboven.pdf

Требования к тестам

При анализе ошибок связанных с многопоточностью использовать минимальные объемы данных:

- **Сократить количество итераций в алгоритмах**
- **Минимизировать размер изображений**

При анализе ошибок памяти использовать максимальные объемы данных.

Использовать полные тесты:

- **Анализируется только исполняемый код**
- **Каждая ветка кода должна исполнится хотя бы один раз**

Jacobi - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отлад

Обозреватель решений

Решение "Jacobi" (проектов: 1

Jacobi

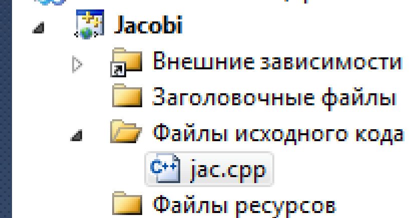
Внешние зависимости

Заголовочные файлы

Файлы исходного кода

jac.cpp

Файлы ресурсов



jac.cpp

(Глобальная

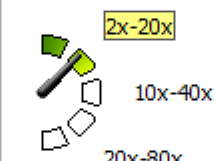
```
/* J
#inc
#inc
#inc
#def
#def
#def

int
doub
doub
FILE
doub
doub
```

150 %

Configure Analysis

Intel Parallel
Inspector 2011

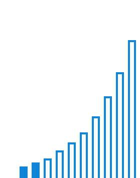
Analysis type: Memory Errors ☑ Always show this dialog box before running memory error analyses



Does my target leak memory?

Does my target have memory access problems?

Where are the memory access problems?



Analysis Time Overhead

Memory Overhead

Does my target leak memory?

- Analysis level: Low (mi1)
- Call stack depth limit: 7

For best results, choose targets:

- Compiled with debug information on, optimization off, and dynamic runtime library selected.
- With small representative data sets (the overhead is proportional to the number of allocation calls)

Press F1 for Help.

Private suppressions: Delete problems

Run Analysis Cancel

Вывод

Показать выход

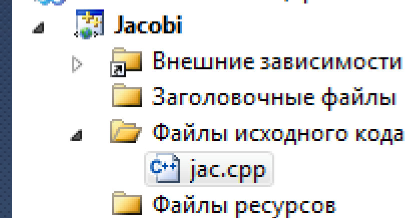
```
1>Построение
1>
1>Затраченное время: 00:00:00.83
========== Построение: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
```

Обозреват... Командны... Окно определения кода Вывод

Jacobi - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отлад

Обозреватель решений



jac.cpp

(Глобальная

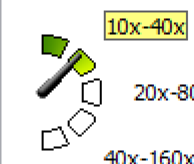
```
/* J
#inc
#inc
#inc
#def
#def
#def

int
doub
doub
FILE
doub
doub
```

150 %

Configure Analysis

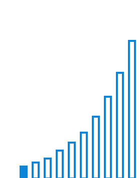
Intel Parallel
Inspector 2011

Analysis type: Threading Errors ☑ Always show this dialog box before running threading error analyses



10x-40x	Does my target have deadlocks?
20x-80x	Does my target have deadlocks or data races?
40x-160x	Where are the deadlocks or data races?

Analysis Time Overhead



Memory Overhead

Does my target have deadlocks

- Analysis level: Low (ti1)
- Call stack depth limit: 1

For best results, choose targets:

- Compiled with debug information on, optimization off, and dynamic runtime library selected.
- With small representative data sets (the overhead is proportional to the number of locking calls)

Press F1 for Help.

Private suppressions: Delete problems

Run Analysis   Cancel

Вывод

Показать выход

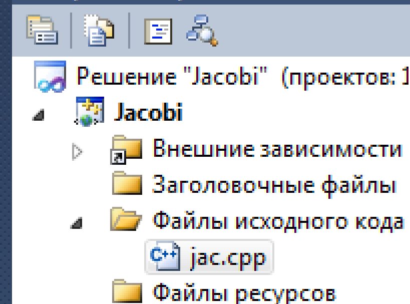
```
1>Построение
1>
1>Затраченное время: 00:00:00.83
========== Построение: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
```

Обозреват... Командны... Окно определения кода Вывод

Jacobi - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отлад...

Обозреватель решений



jac.cpp

(Глобальная

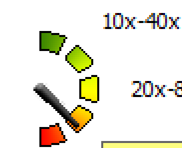
```
/*
#inc
#inc
#inc
#def
#def
#def

int
doub
doub
FILE
doub
doub
```

150 %

Configure Analysis

Intel Parallel
Inspector 2011

Analysis type: Threading Errors ☑ Always show this dialog box before running threading error analyses

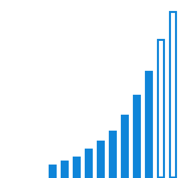


10x-40x Does my target have deadlocks?

20x-80x Does my target have deadlocks or data races?

40x-160x Where are the deadlocks or data races?

Analysis Time Overhead



Memory Overhead

Where are the deadlocks or data races?

- Analysis level: High (ti3)
- Call stack depth limit: 12
- Previous call stacks capture: Yes

For best results, choose targets:

- Compiled with debug information on, optimization off, and dynamic runtime library selected.
- With small representative data sets that run < 20 seconds and use < 200MB of virtual memory.

Press F1 for Help.

Private suppressions: Delete problems ☐ Detect additional problems with stack accesses

Run Analysis Cancel

Вывод

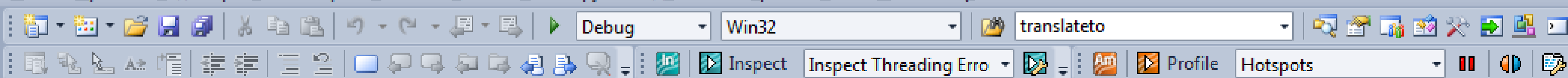
Показать выход...

```
1>Построение...
1>
1>Затраченное время: 00:00:00.83
========== Построение: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
```

Обозреват... Командны... Окно определения кода Вывод

Jacobi - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отладка Рабочая группа Данные Сервис Тест Окно Справка



Обозрев...

Решение "Jacc
Inspector R
Jacobi
r000
r001
r002
r003
r004
Jacobi
Внешни
Заголо
Файлы
jac.c
Файлы

r004ti3 | r003ti3 | r002ti3 | jac.cpp

Threading Errors (level ti3)

Intel Parallel Inspe

Collection Log | **Summary**

Problems

ID		Problem	Sources	Modules	State
P1		Data race	jac.cpp	Jacobi.exe	New
P2		Data race	jac.cpp	Jacobi.exe	New
P3		Data race	jac.cpp	Jacobi.exe	New
P4		Data race	jac.cpp	Jacobi.exe	New

Code Locations

ID	Description	Source	Function	Module
X1	Read	jac.cpp:21	L__main_19__par_loop0_2.27	Jacobi.exe

```
19   #pragma omp parallel for
20       for(i=0; i<=L-1; i++)
21           for(j=0; j<=L-1; j++)
22           {
23               A[i][j]=0.;
```

ID	Description	Source	Function	Module
X3	Read	jac.cpp:23	L__main_19__par_loop0_2.27	Jacobi.exe

```
21           for(j=0; j<=L-1; j++)
22           {
23               A[i][j]=0.;
```

Filters

Severity
Error

Problem
Data race

Source
jac.cpp

Module
Jacobi.exe

State
New

Suppressed
Not suppressed

Investigated
Not investigated

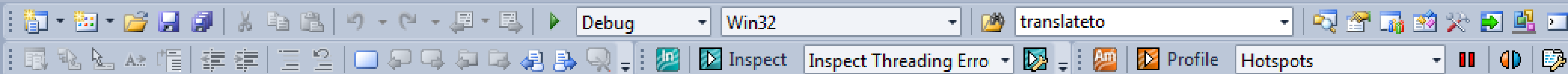
Вывод

Об... | Ко... | Окно определения кода | Вывод

Готово

Jacobi - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отладка Рабочая группа Данные Сервис Тест Окно Справка



Обозрев...

Решение "Jacc
Inspector R
Jacobi
r000
r001
r002
r003
r004
r005
Jacobi
Внешн
Заголо
Файлы
jac.c
Файлы

r005ti3 | r004ti3 | r003ti3 | r002ti3 | jac.cpp

Threading Errors (level ti3)

Intel Parallel Inspe

Collection Log | Summary | Sources

Focus Code Location: jac.cpp:30 - Write

```
28          for(it=1; it<=ITMAX; it++)
29          {
30              eps= 0.;
31  #pragma omp for
32              for(i=1; i<=L-2; i++)
33                  for(j=1; j<=L-2; j++)
34                  {
```

Call Stack

Jacobi.exe!L__main_27__par_

Related Code Location: jac.cpp:35 - Read

```
33                  for(j=1; j<=L-2; j++)
34                  {
35                      eps = Max(fabs(B[i][j]-A[i][j]),eps);
36                      A[i][j] = B[i][j];
37                  }
38  #pragma omp for
39              for(i=1; i<=L-2; i++)
```

Call Stack

Jacobi.exe!L__main_27__par_

Code Locations

ID	Description	Source	Function	Module
X14	HINT: Synchronization allocation site	jac.cpp:19	main	Jacobi.exe
X15	HINT: Synchronization allocation site	jac.cpp:39	L__main_27__par_region1_2.88	Jacobi.exe
X6	Read	jac.cpp:35	L__main_27__par_region1_2.88	Jacobi.exe
X9	Read	jac.cpp:44	L__main_27__par_region1_2.88	Jacobi.exe

Relationships

Read
jac.cpp:35

Вывод

Об... | Ко... | Окно определения кода | Вывод

Готово

Intel Parallel Inspector

inspxe-cl -collect=<string> [-action-option] [-global-option] [--] <target> [<target options>]

<string> gives the analysis type to perform on <target>.

Available analysis types:

Name	Description
mi1	Detect Leaks
mi2	Detect Memory Problems
mi3	Locate Memory Problems
ti1	Detect Deadlocks
ti2	Detect Deadlocks and Data Races
ti3	Locate Deadlocks and Data Races

```
export OMP_NUM_THREADS=4
inspxe-cl -collect=ti3 ./jac
```

Intel Parallel Inspector

inspxe-cl -report=<string> [-action-option] [-global-option]

<string> gives the report type. Supported report types are:

summary

problems

observations

status

Supported report output formats are:

text (default)

csv

xml

inspxe-cl -report=problems

Intel Parallel Inspector

P1: Error: Data race: New

P1.15: Error: Data race: New

jac.c(20): Error X30: Write: Function main: Module a.out

jac.c(20): Error X31: Write: Function main: Module a.out

P2: Error: Data race: New

P2.27: Error: Data race: New

jac.c(36): Error X66: Write: Function main: Module a.out

jac.c(40): Error X67: Read: Function main: Module a.out

P3: Error: Data race: New

P3.24: Error: Data race: New

jac.c(39): Error X57: Write: Function main: Module a.out

jac.c(39): Error X58: Read: Function main: Module a.out

Valgrind

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.

The Valgrind distribution currently includes seven production-quality tools: **a memory error detector**, **two thread error detectors**, **a cache and branch-prediction profiler**, **a call-graph generating cache and branch-prediction profiler**, and **two different heap profilers**.

Helgrind is a thread debugger which finds data races in multithreaded programs (--tool=helgrind).

DRD is a tool for detecting errors in multithreaded C and C++ programs

(-tool=drd).

https://valgrind.org/

Спасибо за внимание!

Вопросы?

Контакты

- **Бахтин В.А.**, кандидат физ.-мат. наук, заведующий сектором, Институт прикладной математики им. М.В.Келдыша РАН

bakhtin@keldysh.ru