Parallel Computing

Academic year – 2020/21, spring semester

Computer science

Lecture 3

Lecturer, instructor:

Balakshin Pavel Valerievich

(pvbalakshin@itmo.ru; pvbalakshin@hdu.edu.cn)

Assistant:

Liang Tingting

(liangtt@hdu.edu.cn)

Goals of parallel programming usage

1. Increasing computational speed.
2. Improving responsiveness of the user interface.
3. Providing high fault-tolerance of application modules when neighboring modules fail.
4. Improving source code structure (reducing dependencies between program modules).

Conditions for successful automatic parallelization

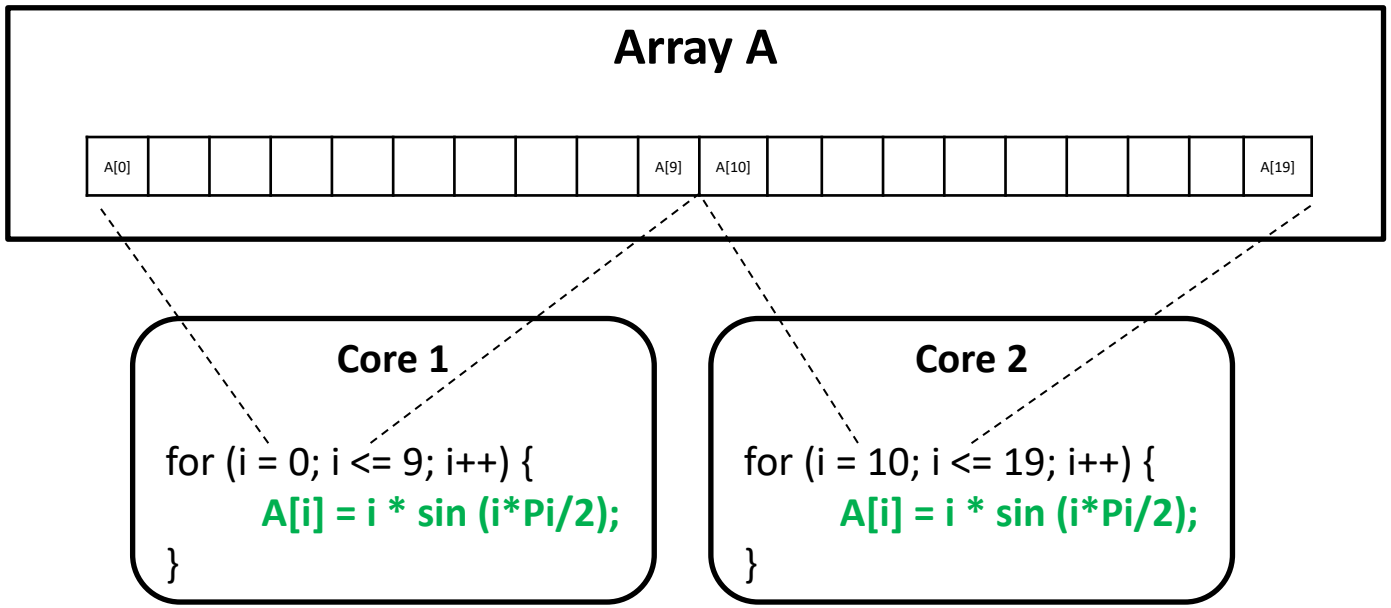
- Only for-cycles (not while, do, etc.) with the number of iterations calculated in runtime.
- No dependencies between different iterations.
- Absence of conditional changes of the variables to be used after the loop inside the loop.
- Sufficient number of iterations and/or sufficient duration of each iteration.
- Absence of function calls within the loop ("side effects", use inline and restricted).

Main approaches to parallelization

- Data parallelization (decomposition).
- Instruction parallelization.
- Parallelization of information flows.

Data parallelization

All cores (threads) execute the same set of instructions at the same time; only the input data for these instructions differ.

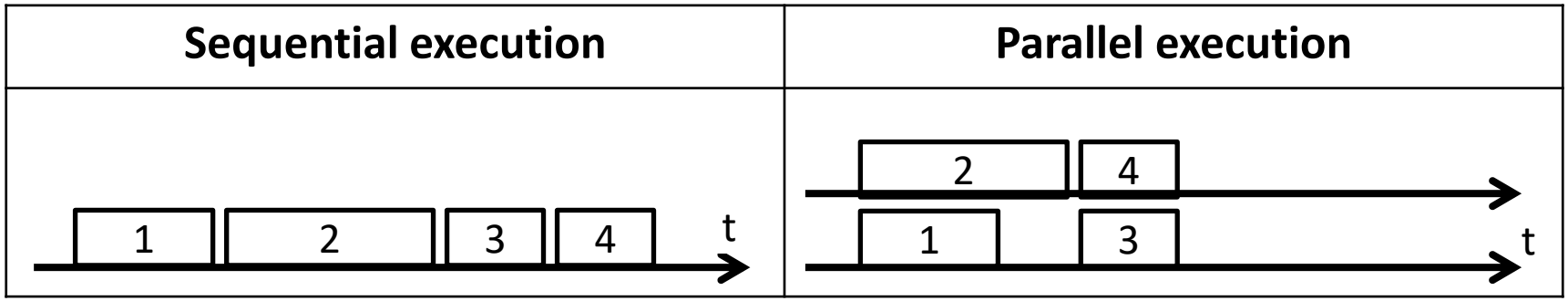


Instruction parallelization

All cores simultaneously execute different unrelated instruction sets; the data sets can both match (blue example) and differ (green example).

A program fragment:

1. Replicate matrices A and B.
2. Reverse the sign of matrix C elements.
3. Find the maximal element of matrix C.
4. Find the minimum element of matrix C.

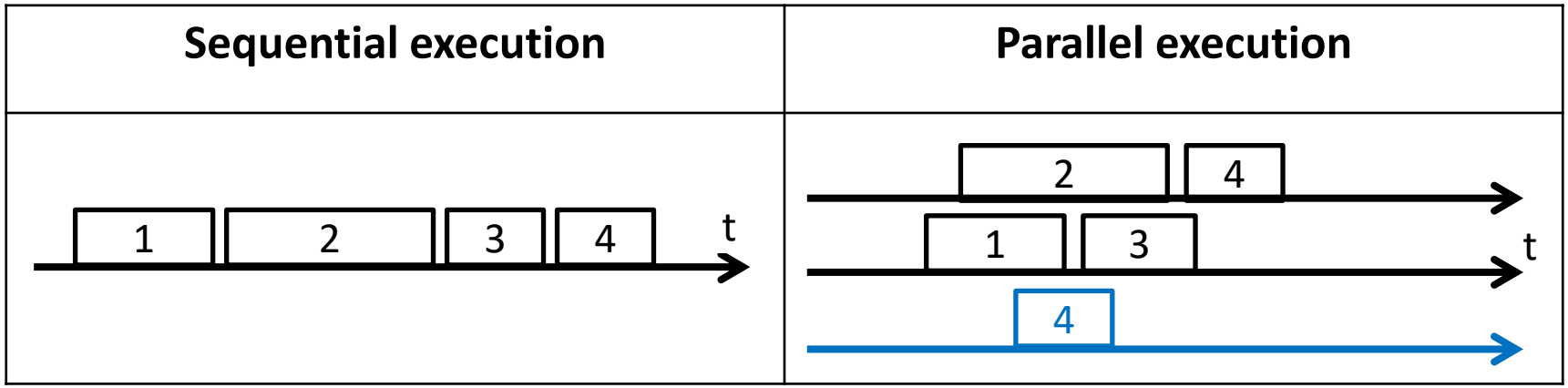


Parallelization of information flows

Cores pass data processing results through the chain to each other (until the core results appear, all downstream cores can't start working).

A program fragment:

1. Multiply matrices A and B, write the result into matrix C.
2. Reverse the sign of matrix C elements.
3. Find the maximal element of matrix C.
4. Find the minimum element of matrix C. (if there is 3rd core)

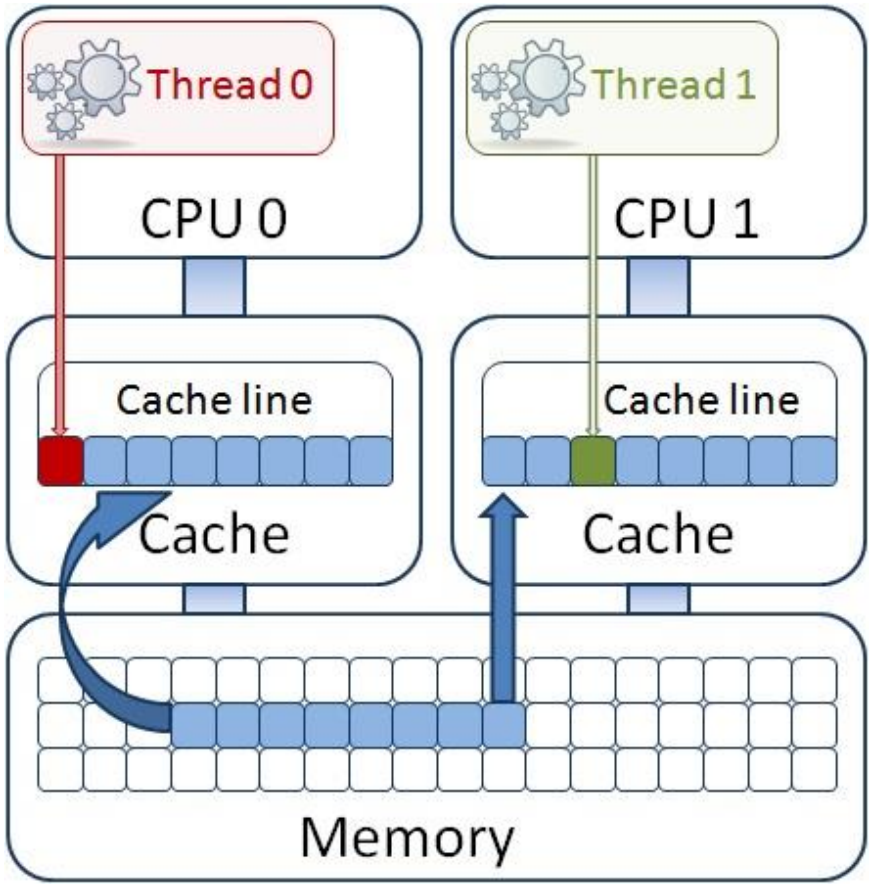


Specifics of paralleling methods

- When paralleling by tasks, there is a high probability of idle time of unloaded cores.
- The data paralleling method is very well scaled but potentially more often it calls False Sharing.
- The method of information flows paralleling has an unrecoverable initial delay.

False sharing of cache lines

The essence of the problem: read and write operations by different threads of different variables located in the same cache string lead to unnecessary waiting for cache string rebooting for the reading stream through the fault of the writing stream.



Solutions:

1) Spacing variables in memory
2) Creating local copies of variables before they are used in the thread

Structure of Lab #1

1. Generate M1
2. Generate M2
3. Map M1 (no dependencies)
4. Map M2 (there is dependency)
5. Merge M1 and M2 into M2
6. Sort M2
7. Bring M2 to one number (a cycle with conditions inside iterations)

Paralleling libraries

- AMD Performance Library (Framewave)
- Intel Integrated Performance Primitives (IPP), Intel Math Kernel Library
- Sun MediaLib
- ATLAS (Automatically Tuned Linear Algebra Software): MATLAB, Mathematica, Octave, …
- …

Conflicts: https://www.agner.org/forum/viewtopic.php?t=6

Hints for Lab #2

1. Download already compiled library (https://sourceforge.net/projects/framewave/files/framewave-releases/Framewave%201.3.1/FW_1.3.1_Lin64.tar.gz/download) or (https://fossies.org/linux/misc/old/FW_1.3.1_Lin64.tar.gz/index_t.html).
2. Unpack the downloaded archive to any convenient directory.
3. Create environment variable `FW_HOME`, containing path to unpacked library.
4. Run the script:

```
FW_LIB=$FW_HOME/lib; for file in $(ls -1 $FW_LIB); do ln -sf
$FW_LIB/$file $FW_LIB/$(echo $file | sed 's/.1.3.1//'); ln -sf
$FW_LIB/$file $FW_LIB/$(echo $file | sed 's/.3.1$//'); done
```

5. Use keys `-I${FW_HOME}` and `-L${FW_HOME}/lib/` to compile (and link).
6. Change `LD_LIBRARY_PATH` to launch the compiled program:

```
export LD_LIBRARY_PATH=$FW_HOME/lib
```

7. Add required header files to your code: `#include <fwBase.h>`

Hints for Lab #2 (2)

```
void merge(struct array_d m1, struct array_d m2) {
    for (int i = 0; i < m2.size; ++i) {
        m2.array[i] = fabs(m1.array[i] - m2.array[i]);
    }
}
```

```
void merge(struct array_d m1, struct array_d m2) {
    fwsSub_64f_I(m1.array, m2.array, m2.size);
    fwsAbs_64f_I(m2.array, m2.size);
}
```

```
void map(struct array_d m1, struct array_d m2) {
    for (int i = 0; i < m1.size; i++) {
        double temp = m1.array[i] / M_E;
        m1.array[i] = cbrt(temp);
    }

    double* restrict temp_arr = malloc(m2.size * sizeof(double));
    for (int i = 0; i < m2.size; ++i) {
        temp_arr[i] = m2.array[i];
    }
    m2.array[0] = map_m2(0.0, m2.array[0]);
    for (int i = 1; i < m2.size; ++i) {
        m2.array[i] = map_m2(temp_arr[i - 1], temp_arr[i]);
    }
    free(temp_arr);

}
```

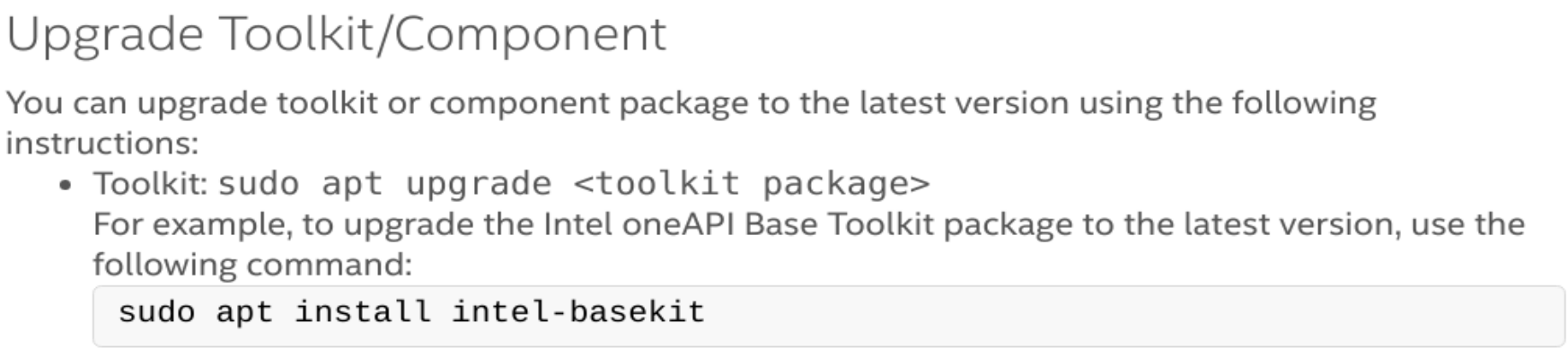
```
void map(struct array_d m1, struct array_d m2) {
    fwsDivC_64f(m1.array, M_E, m1.array, m1.size);
    fwsCbrt_64f_A50(m1.array, m1.array, m1.size);

    double* restrict temp_arr = malloc(m2.size * sizeof(double));
    fwsCopy_64f(m2.array, temp_arr, m2.size);
    fwsAdd_64f_I(temp_arr, m2.array + 1, m2.size - 1);
    fwsLog10_64f_A50(m2.array, m2.array, m2.size);
    fwsPowx_64f_A50(m2.array, M_E, m2.array, m2.size);
    free(temp_arr);

}
```

```
// map
for (int j = 0; j < lenM1; j++) {
    M1[j] = exp(sqrt(M1[j]));
}
```

```
// map
fwsSqrt_64f_I(M1, lenM1);
fwsExp_64f_I(M1, lenM1);
```

Hints for Lab #2 (3)

1. Follow the instructions for your operation system: (https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit/download.html).
2. Download extended version for IPP:



3. Apply environment for ICC (Linux):

```
. /opt/intel/oneapi/setvars.sh
```

5. Use IPP documentation: (https://software.intel.com/content/dam/develop/external/us/en/documents/ipps.pdf)

Hints for Lab #2 (4)

```
void map_M1(float *array, int size){
   for (int i=0; i < size; i++){
      array[i] = exp(sqrt(array[i]));
   }
}
```

```
void map_M1(float *array, int size){
   ippsSqrt_32f(array, array, size);
   ippsExp_32f(array, array, size);
}
```

```
void map_M2(float *array, int size){
   float array_copy[size];

   for (int i=0; i < size; i++){
      array_copy[i] = array[i];
   }

   float past = array_copy[0];

   for (int j=1; j<size; j++){
      array[j] = abs(sin(array_copy[j]+past));
      past = array_copy[j];
   }
}
```

```
void map_M2(float *array, int size){
   float array_copy[size];

   ippsCopy_32f(array, array_copy, size);
   array_copy[0] = 0;

   ippsAdd_32f_I(array_copy, array, size);
   ippsSin_32f_A21(array, array, size);
   ippsAbs_32f(array, array, size);
}
```