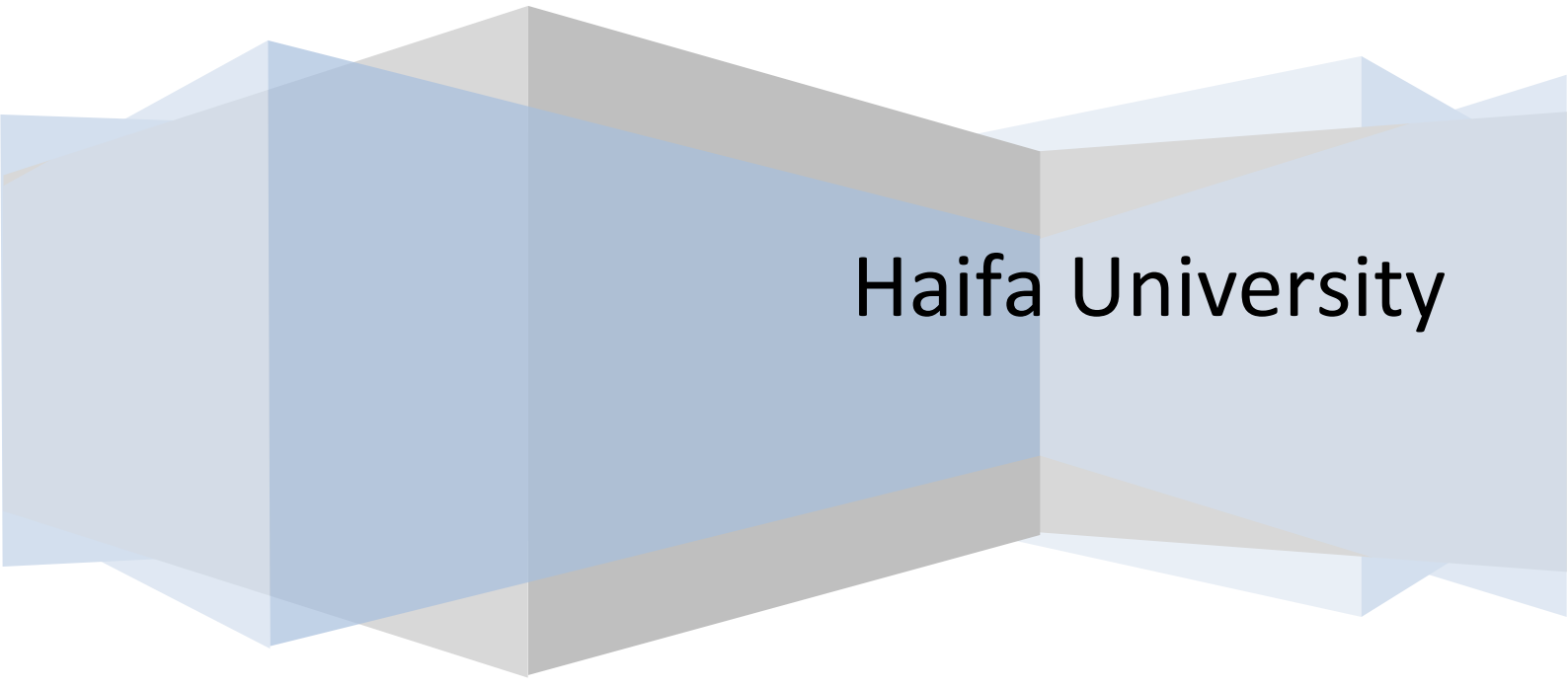
ParC Manual

A quick install-and-run guide

Prof. Yosi Ben Asher, Gil Kulish



Haifa University

ParC User Guide

Table of Contents

PARC USER GUIDE 2

1. PARC VERSION 3
2. PREINSTALLATION REQUIREMENTS 3
3. PARC INSTALLATION INSTRUCTIONS 3
4. COMPILING AND RUNNING A PARC ("P") FILE 3
5. ALTERNATIVELY: COMPILING AND RUNNING A PARC FILE WITH MEMORY INSTRUMENTATION 3
6. CLEANING 4
7. PARC KNOWN ISSUES 4
7.1. PARFOR STATEMENT MUST BE FOLLOWED WITH CURLY BRACKETS ("{""}") 4
7.2. MEMORY ANALYSIS DOES NOT SUPPORT PROGRAMS WHICH PERFORM EXIT. 5
7.3. NO SUPPORT FOR USING ARRAYS WITH FIXED SIZE (INT X[10]) INSIDE THE PARFOR BRACKETS. 5
7.4. CURRENT VERSION DOES NOT SUPPORT GLOBAL VARIABLE DECLARATIONS 5
7.5. WHEN A PARFOR LOOP HAS ENDED, THE LOOP VARIABLE WILL REGAIN IT'S PREVIOUS VALUE. 6
7.6. A PARFOR LOOP VARIABLE CANNOT BE CHANGED INSIDE THE LOOP 6
7.7. THE BOOLEAN CONDITION INSIDE A PARFOR BRACKETS IS VERY STRICT 7
7.8. PARC VARIABLE DECLARATIONS CONFORM C LANGUAGE VARIABLES DECLARATIONS 7
7.9. USE ONLY #INCLUDE WITH `"` AND NOT WITH '<' '>' 7

1. ParC Version

Version 0.1 (Pre-release)

2. Preinstallation requirements

Unix system / Unix virtual machine / CYGWin (installed for CPP development).

Although ParC system is portable, meaning, it should work on windows OS using Visual C IDE, the testing on Visual C IDE have not been conducted lately, so for the time being, a Unix-like environment should be preferred.

2.1. "Flex" installed, and its executables configured to be on the computer's path.

flex homepage, windows version

2.2. "Bison" installed, and its executables configured to be on the computer's path.

bison homepage, windows version

flex and bison might not work when installed into directories that contain white spaces:

for example "c:**program files**\bison". this is a known issue.

Note that instead of configuring bison and flex on the system path, you can change the ***"LEX:=flex YACC:=bison"*** variables, inside the ParC\src\makefile, to point to your flex and bison executables.

3. ParC Installation Instructions

1.1. Extract the zip into a folder, mind that the zip is password protected.
The zip should contain two directories: "ParC" and "pth-2.0.7".

1.2. Execute the command ***"make"*** from the dir ParC.
This may take a few minutes depending on your computer.
To validate that the installation had finished correctly, check that the file ***"compiler.exe"*** exists inside ParC\lib

4. Compiling and running a ParC ("p") file

4.1. Enter "ParC\lib" and select a parfile to compile. Let's assume you selected the file named "new1.p"

4.2. From that library ("ParC\lib") execute the command:

make PARFILE=new1

4.3. You can now execute ***new1***, mind that if you are using a Unix environment, you should execute new1 using the command ***"./new1"***

5. Alternatively: Compiling and running a ParC file with memory instrumentation

Performing "Memory Profiling" - for the scheduler usage.

5.1. Run "***make clean***" from ***ParC\lib*** in any case where you switch from compiling a file with\without memory profiling.
5.2. Enter "ParC\lib" and select a parfile to compile. Let's assume you selected the file named "new1.p"
5.3. From that library ("ParC\lib") execute the command:
make PARFILE=new1 MEM_PROF=TRUE
5.4. Execute new1 to collect information for the scheduler and to create a profile. Next execution of new1 will use the profile collected in the previous round.
After you collected and created the profile, you can recompile new1.p back to the version that does not collect a profile, and run that new version, this will give the best results.

6. Cleaning

In order to clean the parfiles compilation products, use the command:
"make c" ,from the ***ParC\lib*** directory.

Other cleaning instructions:

"***make clean***" from ***ParC\lib***, will perform a dipper clean,
it is recommended to use this cleaning technique if you plan to switch between compiling with and without memory profiler.

"make clean" from **ParC** will clean most of the primary binaries created in the initial installation (all but the ***pth*** binaries)

"make clean_all" from **ParC** will clean all of the binaries created in the initial installation

7. ParC known issues

The following list describes known bugs and limitations which will be resolved in the near future.
Some of these issues might generate compilation error, but some might not, in some cases the program will get stuck or exit with a segmentation fault, or the results will be irregular.

Please read the next section carefully, as it is best you know these open issues by heart.

7.1. Parfor statement must be followed with curly brackets ("{""}")

Although the ParC compiler will not warn you if you write:

```
parfor (i=0;i<5;i++)
        j=i;
```

The code itself will not function correctly; a correct code should look as follows:

```
parfor (i=0;i<5;i++){
        j=i;
}
```

7.2. Memory Analysis does not support programs which perform exit.

Memory Analysis does not support programs which perform exit using "exit" function, or use "return ;" in the function main()

The following code will not support memory profiling (MEM_PROF):

```
Int main(int argc, char** argc){
…
Some code…
return 0;

}
```

7.3. No support for using arrays with fixed size (int x[10]) inside the parfor brackets.

Array of fixed size is an array declared as follows:

```
Int x[10];
```

This will generate an error:

```
int y,x[10];
parfor (x[1]=0; x[1]<2;  x[1]++){
        y++;
}
```

(error: incompatible types in assignment of `int*' to `int[10]')

and this also:

```
int y,x,z[10];
parfor (x=0;x<z[0]; x++){
        y++;
}
```

(error: incompatible types in assignment of `int*' to `int[10]')

But, this will not generate an error:

```
int y,x,z[10];
parfor (x=0;x<2; x++){
        z[0]++; // this is OK
}
```

7.4. Current version does not support global variable declarations.

Meaning - this will not work:

```
int x;
```

```
int main(int argc, char**argv)
{
…
x=4;
}
```

But this will work fine:

```
int main(int argc, char**argv)
{
        int x;
        …
        x=4;
}
```

7.5. When a parfor loop has ended, the loop variable will regain it's previous value.

It's previous value, meaning, it’s value before the loop.
A parfor loop does not change the value of the loop variable, this is counter intuitive.

For example, the following code:

```
Int main(){
int y=0,x=5,z=0;
parfor (x=0; x<100;  x++){
        parfor (y=0; y<100; y++){
                z++;
        }
}

printf("x=%d y=%d\n",x,y);
}
```

Will always print:
x=5 y=0

7.6. A Parfor loop variable cannot be changed inside the loop.

Meaning- the next code is not allowed:

```
Int main(){
int y=0,x=5,z=0;
parfor (x=0; x<100;  x++){
        z++;
        x=4;
}
}
```

7.7. The Boolean condition inside a parfor brackets is very strict.

The Boolean condition inside a parfor brackets must be in the form of: "IDENTIFIER OPERATOR CONSTANT". Nothing else is supported.

And when I write "nothing else is supported", I mean not even "CONSTANT OPERATOR IDENTIFIER". For example:

This will work:
parfor (i=0; i<5; i++){} // <- OK
but this will not work:
parfor (i=0; 5>i ; i++){} //<- NOT OK

7.8. ParC Variable declarations conform C language variables declarations

The emphasis is that variables must be declared at the top of a block, before any other statment takes place:

```
Int main(){
int y=0,x;
x=5;
int z=0;

parfor (x=0; x<100;  x++){
        z++;
}
}
```

This may yield unexpected results.

7.9. Use only #include with `"` and not with '<' '>'

As for this moment, writing #include with <> will not fail-

#include <stdio.h>// this will not work

Use '"' instead:

#include "stdio.h" // this will work ok