Low Power Optimization for GCC Compiler

Submitted by:
D. Rajeswar Rao – IIT2014055
Nishit Gupta – IIT2014502
Sachin Agarwal – IIT2014501
Saurabh Tanwar – IIT2014140

Aim

- The aim of this project is to :
  - Study the effect of various gcc optimization levels on “Execution Time” and “Power consumed”.
  - Develop a patch for the GNU tool-chain that will aim at reducing Power at various Optimization levels.

Optimizing Compiler

Introduction:

An optimizing compiler tries to **minimize** or **maximize** some attributes of an executable computer program.

The most common requirement is to **minimize** the

- **time** taken to execute a program,
- the **amount of memory** occupied and
- the **power** consumed by a program.

Levels of Optimization

– O0 (Level 0)
No optimization (the default); generates *unoptimized* code but has the *fastest compilation time*.

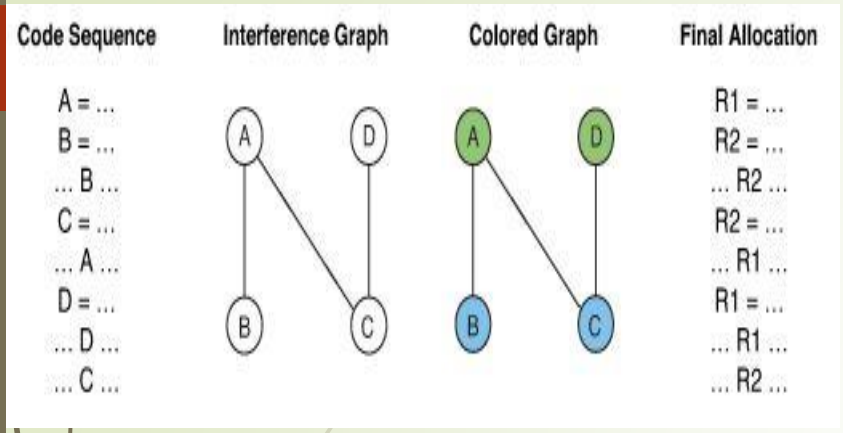
– O1 (Level 1)
Moderate optimization; *optimizes reasonably* well but does not degrade compilation time significantly.
Optimization : global register allocation, dead-code elimination etc.

– O2 (Level 2)
Full optimization; generates *highly optimized code* and has *slower compilation time than O1*.
Optimization : O1 + loop unrolling, thread jumps optimization etc.

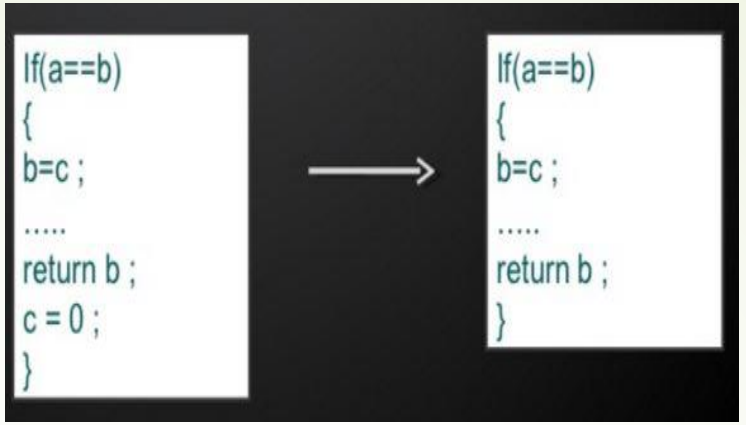
– O3 (Level 3)
Full optimization as in `-O2'; also uses more aggressive automatic inlining of subprograms within a unit (inlining of Subprograms) and attempts to vectorize loops.
Optimization : O2 + prefetching, code motion, scalar replacement etc.

– Ofast (Level 4)
Perform level-3 optimizations plus disregard strict standards compliance.

Optimization Techniques



Global Register Allocation by Graph Coloring

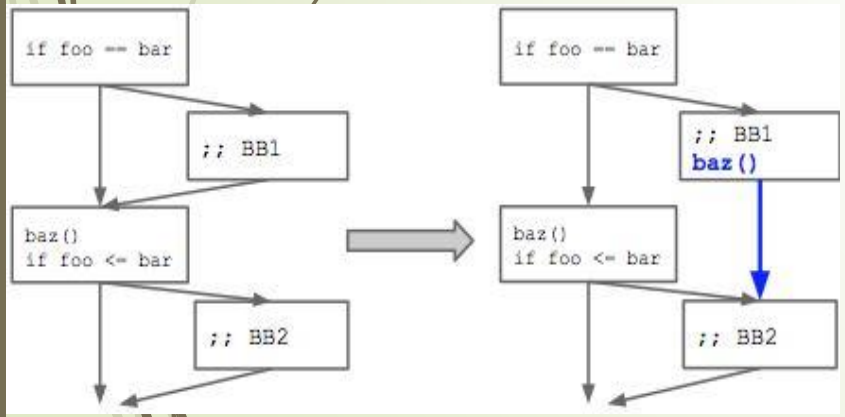


Dead Code Elimination

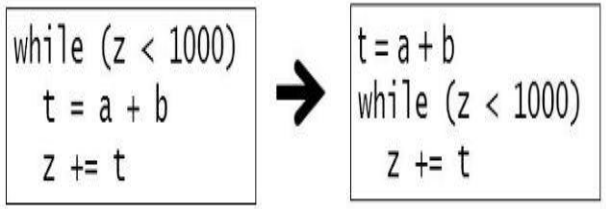
Normal loop	After loop unrolling

`int x;`
`for (x = 0; x < 100; x++)`
`{`
`  delete(x);`
`}`
	`int x;`
	`for (x = 0; x < 100; x += 5)`
	`{`
	`  delete(x);`
	`  delete(x + 1);`
	`  delete(x + 2);`
	`  delete(x + 3);`
	`  delete(x + 4);`
	`}`

Loop Unrolling



Jump Threading



Code Motion

Scalar Replacement

- Before

```
do i = 1, n
    do j = 1, n
        total[i] = total[i] + a[i,j]
    end do
end do
```

- After

```
do i = 1, n
    T = total[i]
    do j = 1, n
        T = T + a[i,j]
    end do
    total[i] = T
end do
```

gcc -O option flag

Set the compiler's optimization level.

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

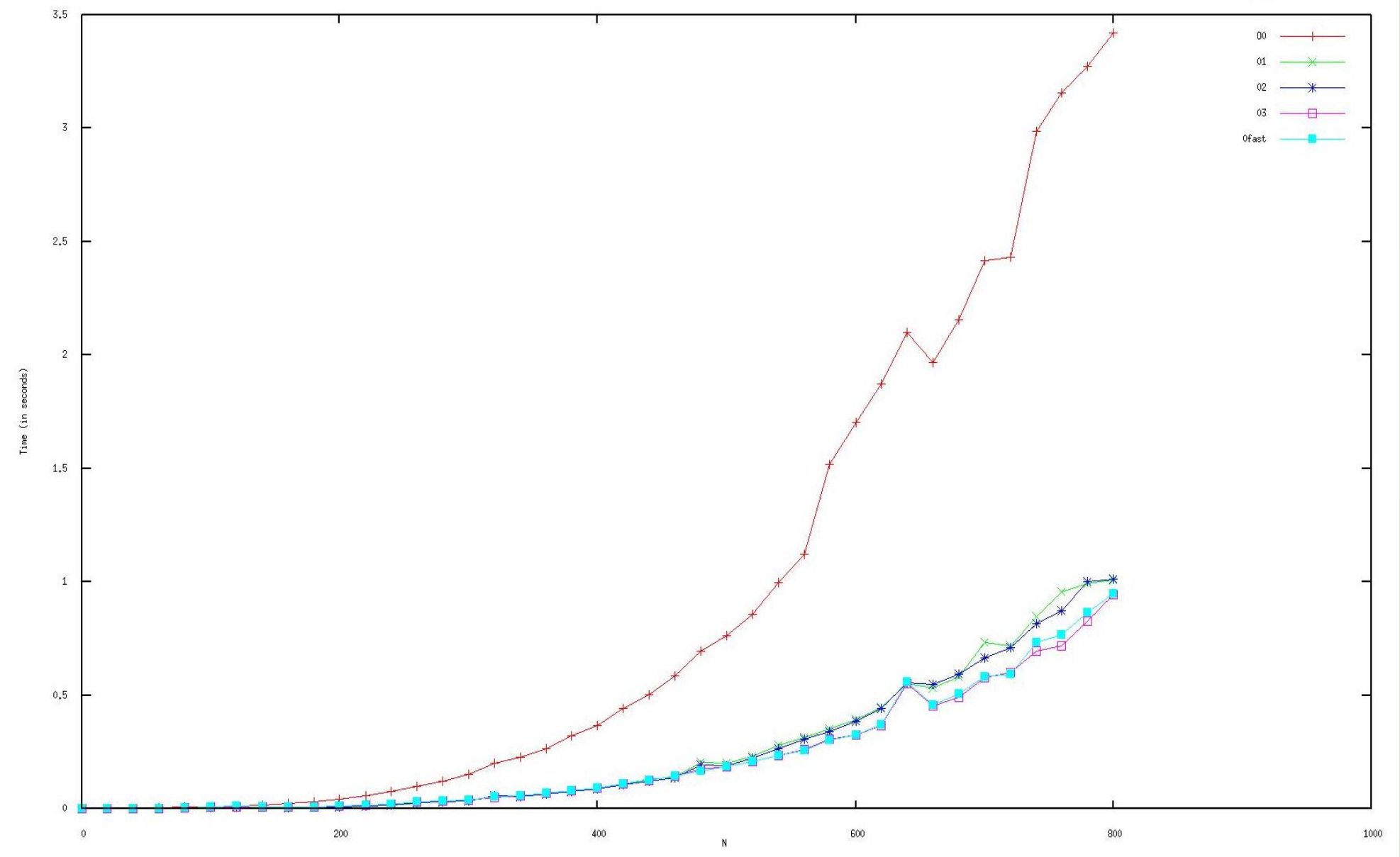
+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more

Syntax

```
$ gcc -Olevel [options] [source files] [object files] [-o output file]
```

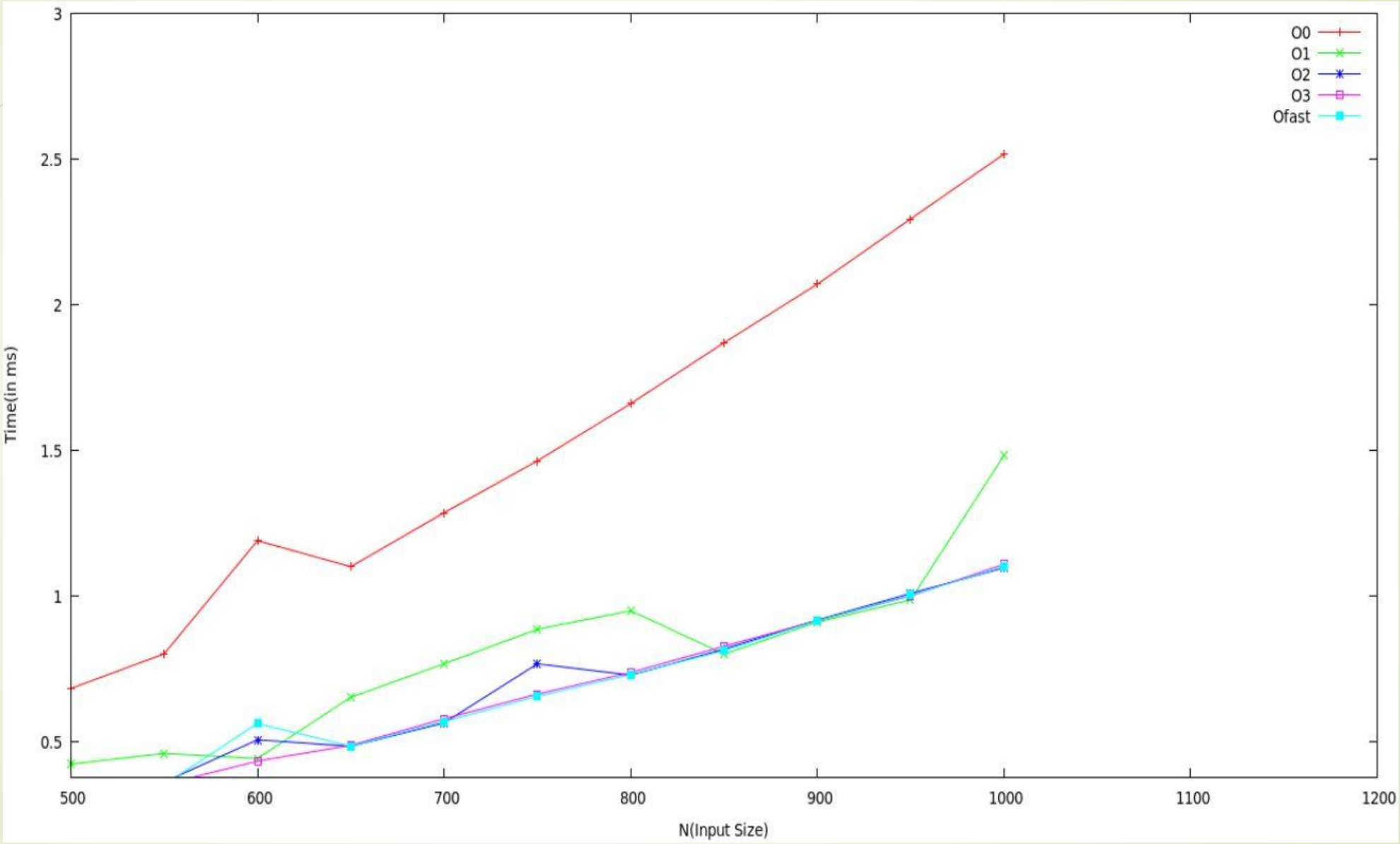
Execution Time Survey

Matrix Multiplication Program : Running Time Complexity – $O(n^3)$

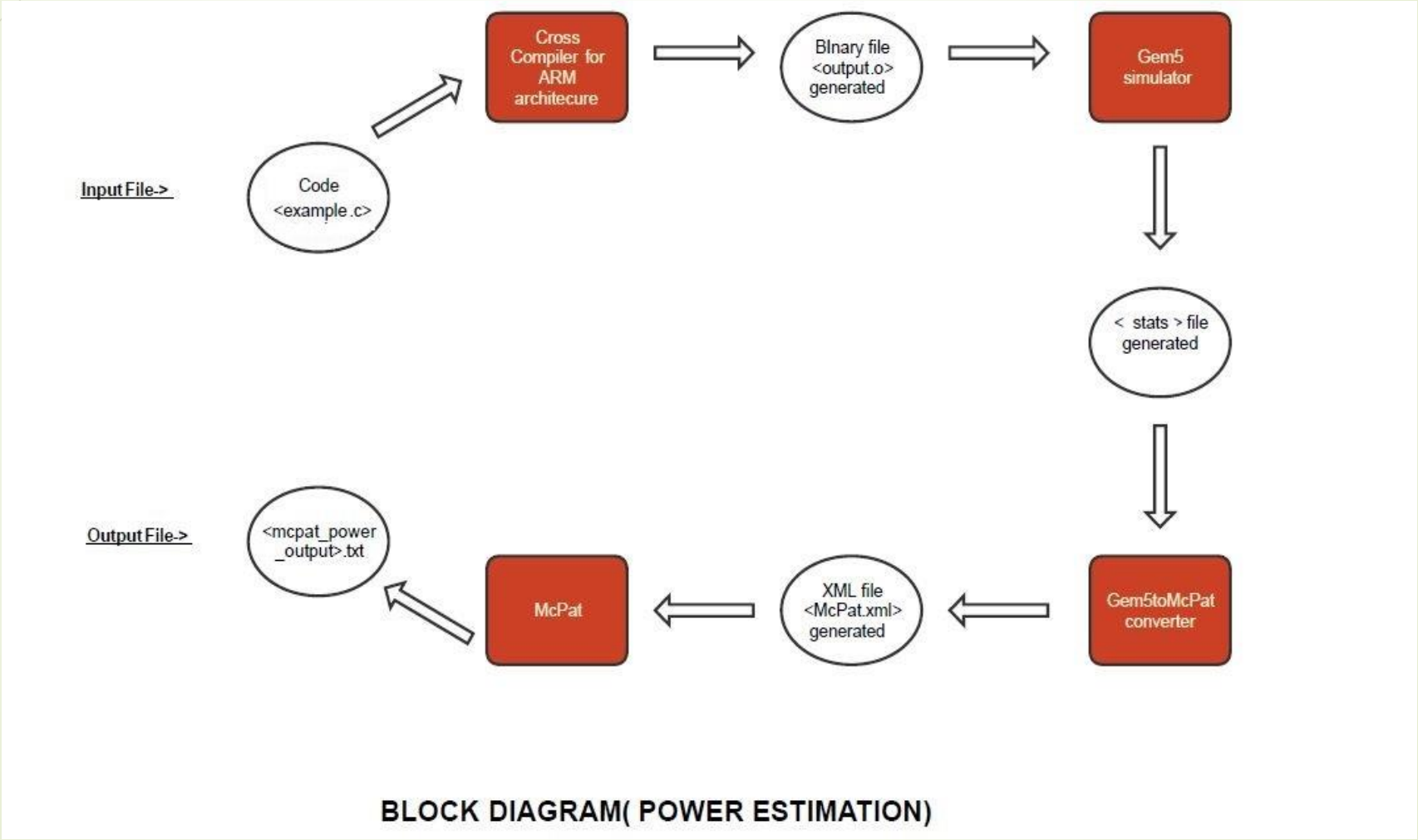


Execution Time Survey

Bubble Sort Program : Running Time Complexity – $O(n^2)$



Steps to Measure Power



BLOCK DIAGRAM(POWER ESTIMATION)

Power Survey

Matrix Multiplication Program : Running Time Complexity – $O(n^3)$

n	O0	O1	O2	O3	Ofast
200	522.784mW	570.949mW	571.1mW	571.1mW	571.1mW
400	511.355mW	517.956mW	517.966mW	517.966mW	517.966mW

Bubble Sort Program : Running Time Complexity – $O(n^2)$

n	O0	O1	O2	O3	Ofast
1000	513.232mW	513.4996mW	513.4996mW	513.4997mW	515.264mW

Conclusion of Analysis

- After Execution Time Analysis on both Matrix Multiplication and Bubble Sort, it was observed that

RUNNING TIME

For Large Values of N:

- **reduces considerably (for larger inputs)** from O0 to O1,
- then reduces relatively less from O1 to O2,
- and similarly for other levels, with Ofast being the best.

For Small Values of N

- **does not reduce considerably** from O0 to O1,
- then reduces relatively less from O1 to O2,
- and similarly for other levels, with Ofast being the best.

Conclusion of Analysis

POWER

- **increases** from O0 to O1,
- then increases relatively less from O1 to O2,
- then very slow increase (or similar) from O2 to O3
- then either remains stable or increases from O3 to Ofast

Therefore, Power consumption **increases** from O0 to Ofast.

Literature Survey

S No.	Author	Paper Title	Year	Crux	Venue
1.	David Branco and P. R. Henriques	Impact of GCC Optimization levels in energy consumptions during C/C++ program execution. [1]	2015	Presenting experimental setup and method followed to measure and compare resources consumed by a program during execution.	2015 IEEE 13th International Scientific Conference on Informatics.
2.	V.Tiwari, Sharad Malik and A.Wolfe	Compilation techniques for low energy : an overview[2]	1994	Used techniques such as Re-ordering instructions to reduce switching and using patterns for Code generations to reduce Power. Conclusion: Conducted an experiment which reduced power upto 40%.	In Low Power Electronics, 1994, Digest of Technical Papers, IEEE Symposium.
3.	Mahmut Kandemir, N. Vijaykrishnan and Mary Jane Irwin	Power aware computing[3]	2002	Focuses on two power aware low-level techniques: 1) Instruction Scheduling for reducing switching acitivity, and 2) Post-compilation relabeling of Register for reducing Power	In Chapter Compiler Optimizations for Low Power Systems
4.	Madhavi Valluri and Lizy K.John	Is Compiling for Performance – Compiling for Powers? [4]	2001	They present a quantitative study where they examine the effect of the standard optimizations levels −01 to −04 on power and energy of the processor. They also evaluate the effect of four individual optimizations on power/energy and classify them as “low energy” or “low power” optimizations.	Springer US, Boston, MA
5.	Ulrich Kremer	Low Power/Energy Compiler Optimizations[5]	2005	Comparison of Power & Energy and Performance Analysis and concluded that both of them are different strategies and one can not be a by-product of the other.	Low-Power Electronics Design, CRC Press, 2005

Motivation

- As suggested from literature and the survey conducted, the focus of compiler optimization techniques have been on :
  - **reducing execution time** of program.
  - and power reduction has been considered a by product of it.

  i.e. Reducing Power was not taken into consideration during the Optimization
- As a result, compiler optimizations for low power have remained mostly unexplored.
- Hence, there is a need for work in finding out **low power compilation techniques** and study their effect.

Problem Statement

- Exploring various compiler optimization techniques (GCC) and study their effect on power optimization on ARM Architecture.
- Develop low power compiler techniques as patches for gnu tool chain.

Power Optimization

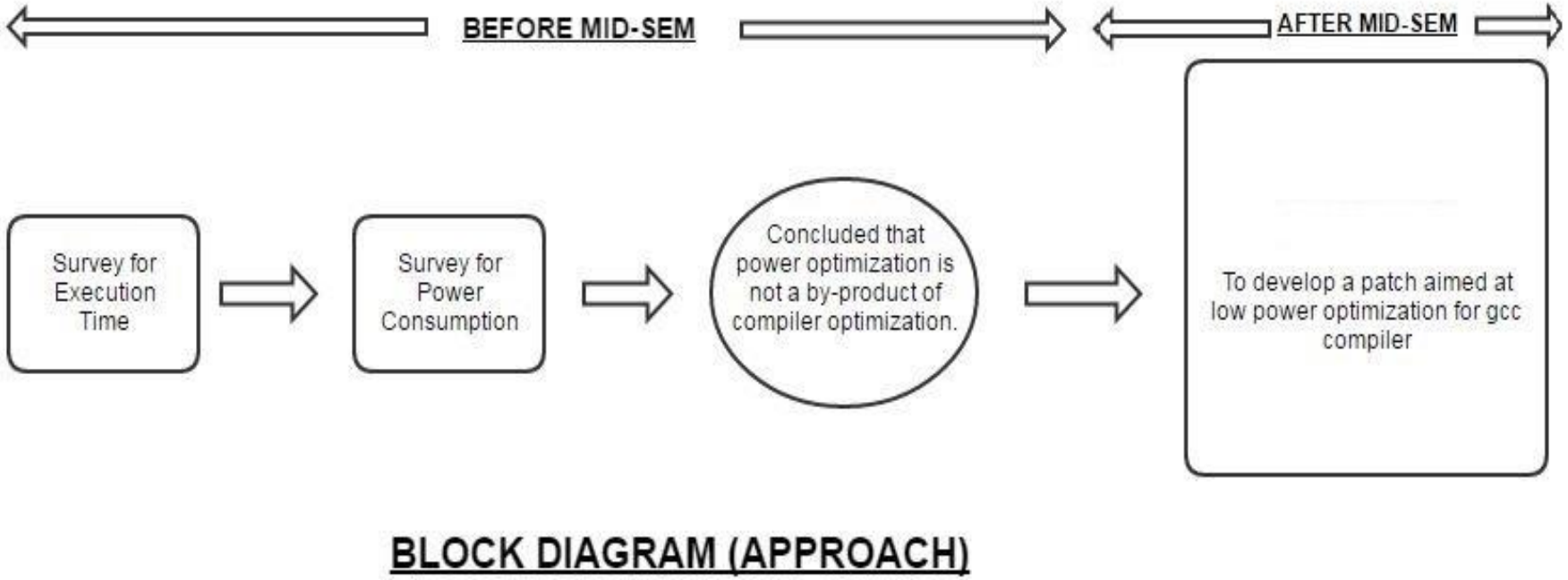
- Power consumption optimizations for embedded systems targets at reducing the power during execution thus making systems much power efficient and less heat generation, and the research on this is growing every year. Important approaches that can be named include:
  - **Dynamic frequency and voltage scaling (DVS)** - idea of DVS is to change the voltage of the power supply unit in some places (called power management points, PMPs) during program execution, such that energy consumption decreases but the needed functionality is still maintained.

Power Optimization (cont.)

- **Tuning of memory optimizations**

  Memory subsystem is considered one of the most energy-consuming parts of embedded devices. Hence, minimization of memory accesses and optimization of cache behavior is an important approach for lowering power consumption.

Work Flow



BLOCK DIAGRAM (APPROACH)

Future Goals

- To Study Various Power Optimization Techniques.
- To Develop a patch for the GNU tool-chain that will aim at reducing Power at various Optimization levels.
- To ensure/prove that the functionality of the program is conserved after applying optimization techniques.