Parallel Analytical Placement

A Project Report

submitted by

NUMAAN A

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

(ELECTRICAL ENGINEERING)

AND

MASTER OF TECHNOLOGY

(MICROELECTRONICS AND VLSI DESIGN)

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2013

THESIS CERTIFICATE

This is to certify that the thesis titled **Parallel Analytical Placement**, submitted by **Numaan A**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Shankar Balachandran
Research Guide
Associate Professor
Dept. of Computer Science and Engg
IIT-Madras, 600 036

Prof. Nitin Chandrachoodan
Research Co-Guide
Associate Professor
Dept. of Electrical Engg
IIT-Madras, 600 036

Place: Chennai

Date: 14^{th} June 2013

ACKNOWLEDGEMENTS

I would like to thank Prof. Shankar Balachandran for giving me the freedom to pursue my ideas while still ensuring that I was on the right track. Without his constant support, I would not have had the confidence to attempt few bold approaches and hence this project would have turned out very differently.

I am immensely grateful towards my lab-mates Abishek, Bharath, Bhargava, Prasad, Siddharth, Sudharshan and Vignesh for their undying support and comradeship, without which this project wouldnot have turned out the way it has. Their ideas and contributions have been instrumental in getting me out of various stagnant phases of this project. I would also like to thank the lab in-charges of DSDL for providing me with the necessary infrastructure in terms of workspace and other facilities.

I would like to extend my deep gratitude to the HPCE team of IIT Madras for giving me unrestricted access to the Virgo Supercluster and Libra GPU Cluster which were instrumental during the development and benchmarking stages of this project.

Thanks are due to my wing-mates Suraj, Gunjan, Aksh, Naseef, Siddarth, Nikhil, Swostik, Ashish, Dilraj, Nishaanth, Sashikanth, Akshay, Rishab, Vinay, Abhitesh without whom my stay at IITM wouldn't have been half as enjoyable an experience as it was. I would cherish these experiences for the rest of my life. I would also like to thank Neeraja, Pooja and Deric for undying love and belief in me.

Finally, none of this would have been possible without the support and encouragement of my parents, brother, sister and sister-in-law. Their unwavering faith in me has been a source of great motivation and continues to remain so.

ABSTRACT

KEYWORDS: Analytical Placement; Electronic Design Automation; Kraftwerk; Parallel; Multicore; CUDA

With rapid scaling of semiconductor manufacturing technology in the recent years, the complexity of integrated circuits has increased drastically leading to millions and billions of components in one single chip. This increasing trend require major improvements in physical design automation to maintain the current pace of innovation. Modern VLSI design flows require considerable effort and time in physical layout, where transistor locations affect nearly all downstream optimizations. However, despite massive improvements in algorithms developed in academia and industry, current placement algorithms leave room for improvement both in quality and speed.

The goal of the placement problem is to position movable components in a fixed area such that no components overlap with each other and some cost metric like overall interconnect wirelength or timing, is optimized. Although it is a classical problem, many modern design challenges have reshaped this problem. As a result, the placement problem has attracted much attention recently, and many new algorithms have been developed to handle the emerging design challenges. Modern placement algorithms can be classified into three major categories: simulated annealing, min-cut, and analytical algorithms. According to the recent literature, analytical algorithms typically achieve the best placement quality for large-scale circuit designs. In this thesis, therefore, we shall select a leading analytical placer, Kraftwerk2, and give a systematic and comprehensive survey on the essential performance bottlenecks and issues in analytical algorithms, and propose ways to improve the overall performance by employing parallel optimization techniques to overcome its computational bottlenecks without making major changes in the overall algorithm itself.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

FPGA	Field Programmable Gate Array
IC	Integrated Circuit
HPWL	Half Perimeter Wirelength
B2B	Bound2Bound
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
MKL	Math Kernel Library
PARDISO	Parallel Direct Solver
CG	Conjugate Gradient

CHAPTER 1

Introduction

1.1 Electronic Design Automation

In the current digital era and electronic devices play a major part in our daily lives. Integrated circuits (ICs) are at the heart of all these devices. MP3 players, mobile phones, laptops, tablets, cars, and even household appliances have high number of integrated circuits in them. And each of these integrated circuits can be made up of millions or billions of individual transistors. Due to this massive complexity of integrated circuits, they can only be designed by algorithms run on computers or on other dedicated hardware like FPGAs. Such algorithms used to design integrated circuits fall under the broad domain of electronic design automation (EDA). Modern EDA problems are very tough because we need to handle these large-scale designs which are growing in scale and complexity every year. Hence, fast and efficient algorithms are necessary for the EDA of future circuits.

1.2 VLSI Design Flow

A broad overview of the electronic design automation process is shown in Figure 1.1. The first step of EDA is to specify the circuit. Here, the main features like performance, functionality, and physical dimensions are defined. After this, the circuit is modelled at system level using a hardware description language like VHDL or Verilog. The next step is logic synthesis, which first transforms the behaviour description of the circuit into a register transfer description. Based on this register transfer model, the gate level description is constructed. After logic synthesis, the circuit is simulated, and various specifications are verified. If the specifications are not met, the previous logic synthesis step is done again. If specifications are met, layout synthesis is done next. During layout synthesis, the placement of gates, and routing of nets are done. Floorplanning

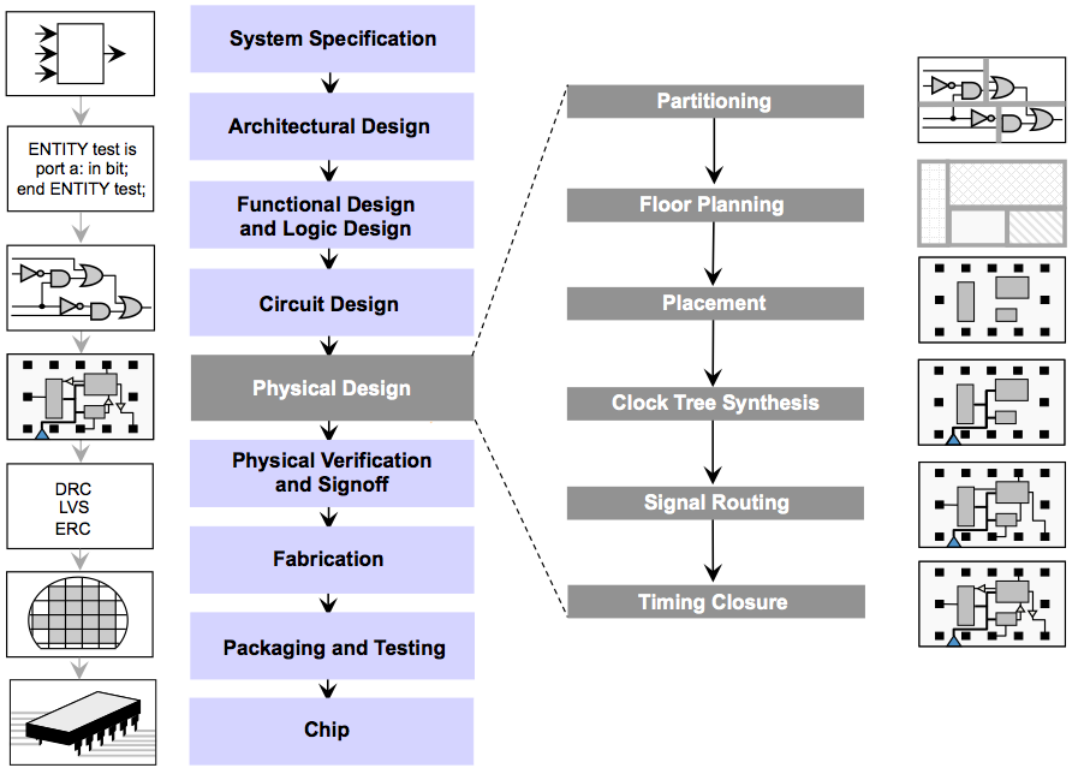

Figure 1.1: VLSI Design Flow

is done to position the I/O pins, and to determine the dimensions of macros. During placement, various design rules are considered, like minimal interconnect between the gates. After placement, net routing is done, after which the entire circuit is described only in terms polygons. At this stage, the circuit is simulated and verified again to check if specifications are met. If not, the EDA is repeated from previous steps. At the end of EDA, the lithography masks are created, and the circuit is fabricated using these masks.

1.3 Placement

Placement is one important step of VLSI design flow, which highly affects the quality of a circuit. The input to the placement algorithm consists of individual gates or bigger macros called modules, interconnected by nets. The goal of the placement step is to determine the positions of these modules, while meeting different objectives and constraints. The fundamental constraints are that the modules do not overlap, and that all modules are located within the chip area. Depending on the type of circuit placed, the modules may be required to align to predefined rows or a grid. Usually, the main ob-

jective of placement is to minimize the total wirelength, i.e., to minimize the sum of the lengths of all nets. This objective is used because with a minimal wirelength, the circuit is easy to route, the maximal clock frequency is high, and the power consumption is low.

Although the placement problem sounds easy, it is a combinatorial problem, which is known to be a NP-complete problem (Donath, 1980). Hence, there exists no algorithm up to date, which solves the problem optimal with polynomial runtime complexity. In the extreme case, a brute-force approach has to be implemented which checks all feasible placements in order to find the optimal placement. With millions of modules, billions of feasible placements, the runtime is not practicable.

Modern placement algorithms can be classified into three major types.

- **Simulated Annealing Based Placement**

  This type of placers attempts to find an optimal placement by perturbing module positions based on simulated annealing. Hence they can adopt various objectives with little modification to the implementation. For smaller designs, a good placement can be achieved due to the small solution space. However, the module perturbation may not be scalable on large-scale circuits. Few placers that use this approach are Dragon (Taghavi *et al.*, 2005) and TimberWolf (Sechen and Sangiovanni-Vincentelli, 1985).

- **Min-Cut Placement**

  Min-cut placement algorithms recursively partitions the circuit and chip region, and then assign sub-circuits into sub-regions in a multilevel approach. There algorithms are usually very efficient and scalable. Min-cut partitioning tries to minimize the expected wirelength between sub-regions by minimizing the number of cuts between sub-circuits. It is also harder to handle whitespace in the earlier levels of the multilevel process. More importantly, the hierarchical approach of solving each subproblem independently might lack the global information for the interaction among different subregions, thus limiting the solution quality. Example min-cut placers are Capo (Adya *et al.*, 2004*a*) and NTUplace (Chen *et al.*, 2005).

- **Analytical Placement**

  Analytical placement formulates the placement problem as mathematical programming composed of an objective function and a set of placement constraints, and then optimizes the objective through analytical approaches. From recent literature and the ISPD placement contests, it can be seen that analytical placement can achieve better placement quality for large-scale circuit designs. Few examples are Kraftwerk2 (Spindler *et al.*, 2008) and mPL6 (Chan *et al.*, 2006), among many others.

It is clear from recent literature and industry driven contests like ISPD, that analytical placers have a distinct advantage over other types of placers in terms of solution quality, consistency and scalability.

The rest of this thesis is organized as follows. Chapter 2 introduces the basic structure of the analytical placement algorithm. Chapter 3 explains in the details the Kraftwerk2 (Spindler *et al.*, 2008) algorithm which is specific analytical placement algorithm we have selected to parallelize and improve in terms of runtime. Chapter 4 explains our efforts in parallelization and optimization of Kraftwerk2. Finally, conclusions are given in Chapter 5.

CHAPTER 2

Analytical Placement

2.1 Basics of Analytical Placement

As mentioned earlier, placement is the process of determining the locations of circuit devices on a fixed die such that no devices overlap with each other and some cost metric (e.g., wirelength) is optimized. Most modern analytical placers consist of the following three major steps:

1. **Global placement** computes the best position for each module to minimize the predefined cost (e.g., wirelength) while ignoring some placement constraints like module overlap. Global placement is generally considered the most important step, due to its crucial impact on the overall placement quality.
2. **Legalization** removes all overlaps among modules.
3. **Detailed placement** further improves the legalized placement solution in an iterative manner by rearranging a small group of modules in a local region.

These three steps are explained in details in the following section.

2.2 Global Placement

During global placement, the circuit is modelled by a hypergraph $H = (V, E)$. Let vertices $V = v_1, v_2, ..., v_n$ represent modules, and hyperedges $E = e_1, e_2, ..., e_m$ represent nets. Let x_i and y_i be the x and y coordinates of the center of module v_i, respectively. The typical objective of global placement is to minimize its wirelength, and a fundamental constraint is to avoid any cell overlap. The global placement problem can be formulated as a constrained minimization problem as follows:

min $W(V, E)$

s.t. no overlaps among modules,

where $W(V, E)$ is the wirelength function. The minimization problem consists of two parts, wirelength estimation and overlap reduction. The wirelength models needed for wirelength estimation are explained in Section 2.3, and overlap reduction techniques are explained Section 2.4. Methods used to integrate the aforementioned parts are explained in Section 2.5. Finally, few cost optimization techniques are explained in Section 2.6.

2.3 Wirelength Models

The wirelength of a net $e \in E$ is usually defined by its total half-perimeter wirelength (HPWL) as follows:

$$W(V, E) = \sum_e (\max_{v_i, v_j \in E} |x_i - x_j| + \max_{v_i, v_j \in E} |y_i - y_j|) \tag{2.1}$$

$$= \sum_e (\max_{v_i \in E} x_i - \min_{v_i \in E} x_i + \max_{v_i \in E} y_i - \min_{v_i \in E} y_i) \tag{2.2}$$

$$= \sum_e (L_{e,x} + L_{e,y}) \tag{2.3}$$

However, $W(V, E)$ is not differentiable, it is hard to find its minimum value. Hence, it is necessary to use a continuous differentiable function to approximate the HPWL. Few popular smooth wirelength approximations are explained in the following subsections.

2.3.1 Quadratic Model

Quadratic wirelength model is the halved sum of the quadractic euclidean length between all two-pin connections. Hence wirelength can be represented as

$$L_{e,x} = \frac{1}{2} \sum_{i=1}^{P} \sum_{j=i+1}^{P} w_{x,ij} (x_i - x_j)^2 \tag{2.4}$$

where P represents the number of pins in net n. The half scaling factor is used to have a simpler derivative form. Multi-pin nets are often modelled by the clique net model or the star net model to fit the two-pin connection model. The clique model considers all possible two-pin connections of a net, while the star net model introduces an additional star pin per net and connects each pin of the net to the star pin. The clique model is equivalent to the star net model in the quadratic cost, if the clique cost is scaled with $1/P$ (Viswanathan and Chu, 2005).

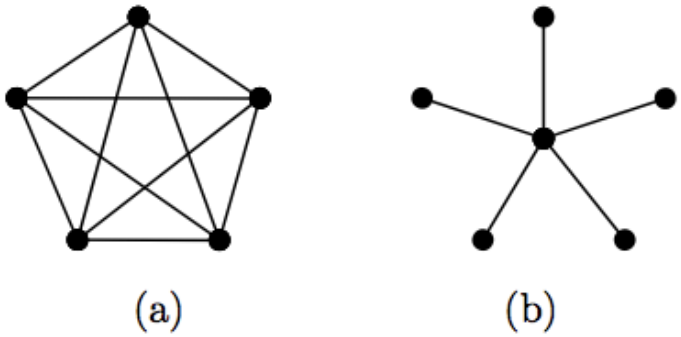

Figure 2.1: Two models for a five-pin net. (a) The clique model. (b) The star model.

The net weight $w_{x,ij}$ is used to adjust the quadratic objective to approximate the linear objective (HPWL). For example, Gordian-L (Sigl *et al.*, 1991) uses the following formula to determine the x-component weight for the approximation:

$$w_{x,ij}^{GordianL} = \frac{1}{P}\frac{2}{P}\frac{4}{|x_i - x_j|} \tag{2.5}$$

The first term $1/P$ adjusts the clique model to the star net model. The second term $2/P$ adjusts the number of connections of the clique to the number of connections in the corresponding spanning tree. The third term $1/|x_i - x_j|$ linearizes the quadratic distance between two pins.

2.3.2 Bound2Bound Model

Regardless of the $w_{x,ij}$, the clique net model has a high approximation error between the total length of the clique net and the HPWL. The problem of the clique model is that its inner connections add to the clique length, which are otherwise not considered for HPWL, as shown in Figure 2.2(a). In this figure, the boundary pins are those with the

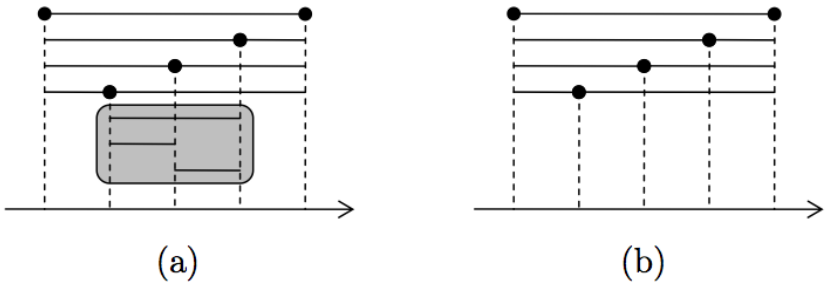

Figure 2.2: The clique net model and the Bound2Bound net model. (a) shows inner connections that are not added in the HPWL calculation. (b) B2B net model with inner connections removed

smallest/largest coordinates, and other pins are inner pins. There are three connections only connecting to inner pins, but these connections are ignored in the HPWL metric.

The Bound2Bound net model removes all inner two-pin connections, as shown in Figure 2.2(b). The net weight $w_{x,ij}^{B2B}$ of the Bound2Bound net model is given below

$$w_{x,ij}^{B2B} = \begin{cases} 0 & v_i, v_j \in \text{inner pins} \\ \frac{2}{P-1}\frac{1}{|x_i - x_j|} & \text{else.} \end{cases}$$

With this connection weight, the quadratic wirelength function in Equation 2.4 exactly matches the HPWL (Spindler *et al.*, 2008).

$$L_{e,x} = \frac{1}{2}\sum_{i=1}^{P}\sum_{j=i+1}^{P} w_{x,ij}^{B2B}(x_i - x_j)^2 \tag{2.6}$$

$$= \max_{v_i \in e} x_i - \min_{v_i \in e} x_i \tag{2.7}$$

2.3.3 LSE Model

To accurately approximate and to smooth the HPWL, logarithm-sum- exponential (LSE) approximation of the max/min function is prevailing in recent placers, such as APlace (Kahng and Wang, 2005), mPL6 (Chan *et al.*, 2006), and NTUplace3 (Chen *et al.*,

2008). The HPWL of a net $e \in E$ can be approximated by using LSE as follows:

$$LSE_e = \gamma \left(\log \sum_{v_k \in e} \exp \frac{x_k}{\gamma} + \log \sum_{v_k \in e} \exp \frac{-x_k}{\gamma} + \log \sum_{v_k \in e} \exp \frac{y_k}{\gamma} + \log \sum_{v_k \in e} \exp \frac{-y_k}{\gamma} \right) \tag{2.8}$$

When γ approaches zero, the LSE wirelength is close to the HPWL (Naylor *et al.*, 2001).

$$\lim_{\gamma \to 0} LSE_e = HPWL_e \tag{2.9}$$

However, due to the computer precision, we can only choose a reasonably small γ to avoid any arithmetic overflow during the implementation. In particular, LSE_e is differentiable, and thus it serves as a good approximation to $HPWL_e$, in terms of precision as well as computation.

2.3.4 Lp-norm Model

Another good smoothing method of the HPWL is the Lp-norm approximation:

$$Lpnorm_e = \left(\sum_{v_k \in e} x_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} x_k^{-p} \right)^{-\frac{1}{p}} + \left(\sum_{v_k \in e} y_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} y_k^{-p} \right)^{-\frac{1}{p}} \tag{2.10}$$

The use of parameter p is similar to γ in LSE model. When p is close to zero, the Lp-norm model gives a good approximation to the HPWL.

$$\lim_{p \to \infty} Lpnorm_e = HPWL_e \tag{2.11}$$

Due to the computer precision, similarly, we can only choose a reasonably large p to prevent any arithmetic overflow during the implementation. The authors in (Chan *et al.*, 2006) compared the LSE and Lp-norm models and concluded that the LSE model usually outperforms the Lp-norm one in terms of HPWL.

2.4 Overlap Reduction

The second step for analytical placement is the reduction of overlap between the modules and distribute them evenly within the placement area. Few popular overlap reduction techniques are described in the following subsections.

2.4.1 Partitioning

Partitioning decomposes the circuit into smaller subcircuits and assigns those partitioned subcircuits to appropriate subregions. The movement of each cell is constrained within their subregion and hence the amount of overlaps can be reduced. Partitioning-based overlap reduction is done in two stages, the partitioning stage and the refinement stage. In the partitioning stage, with a given initial placement, the circuit is partitioned and assigned to subregions. In the refinement stage, various heuristics are used to improve the partition quality. If the resulting partitions are not able to fit within the subregion, the partition is refined by adjusting the size of each partition. This can be done by formulating a transportation problem (Brenner and Struzyna, 2005) to assign modules to subregions while minimizing displacement.

2.4.2 Cell Shifting

Another possible method to reduce cell overlaps is to spread cells through cell shifting. This was first proposed by Viswanathan and Chu in (Viswanathan and Chu, 2005). The basic idea is to distribute modules over the placement region while preserving their relative order obtained from initial placement. During cell shifting, the placement region is divided into equal-sized bins first and an adjusted bin structure is constructed according to the current utilization of each bin. Then, every module is moved based on the linear mapping from the initial bin structure to the adjusted one.

2.4.3 Min Cost Flow

This was first proposed in (Agnihotri and Madden, 2007). A physical clustering is first performed to cluster nearby cells together and then the placement region is partitioned into uniform subregions, and a minimum cost flow algorithm is used to assign clusters into the corresponding subregions. After the construction of clusters and subregions, the best assignment of the clusters to the subregions is found by formulating and solving a minimum cost flow problem.

2.4.4 Diffusion

Use of physical diffusion process to reduce overlap was introduced in (Ren *et al.*, 2007). The physical diffusion process is driven by the gradient of concentration. A velocity function is used to determine movement of modules from initial placement to final optimal placement. The velocity is determined by the amount of density and the local density gradient. The diffusion equations are discretized so that they can be solved for every time-step. The placement region is divided into equal-sized bins and its density is calculated based on modules that fall within each bin. The discretized diffusion equations are solved on the density grids and modules are moved from higher density areas to lower density ones with decreasing velocity while the module reaches its optimal position.

2.4.5 Density

Density based approach is one of the most popular methods to reduce overlap. It is adopted by various placers, such as NTUplace3 Chen *et al.* (2008), APlace (Kahng and Wang, 2005), Kraftwerk (Spindler *et al.*, 2008), FDP (Vorwerk *et al.*, 2004) and mFAR (Hu *et al.*, 2005). The placement region is first discretized into uniform bins. The density function for bin b is

$$D_b(x, y) = \sum_{v \in V} P_x(b, v) P_y(b, v) \tag{2.12}$$

where P_x and P_y are the overlap functions of bin b and module v along the x and y directions. The new overlap constraint becomes

$$D_b(x, y) \leq M_b \text{ for each bin } b \tag{2.13}$$

where M_b is the maximum allowable area of modules in bin b. As this density $D_b(x, y)$ is neither smooth nor differentiable, it is hard to optimize it easily. Hence various smoothing techniques are adopted to make $D_b(x, y)$ differentiable. The popular ones are described below.

- **Bell-Shaped Smoothing:** Bell-shaped smoothing uses the following function to smooth the overlap function P_x

$$p_x(b, v) = \begin{cases} 1 - ad_x^2 & 0 \leq d_x \leq \frac{w_v}{2} + w_b \\ b(d_x - \frac{w_v}{2} - 2w_b)^2 & \frac{w_v}{2} + w_b \leq d_x \leq \frac{w_v}{2} + 2w_b \\ 0 & \frac{w_v}{2} + 2w_b \leq d_x \end{cases}$$

  where

$$a = \frac{4}{(w_v + 2w_b)(w_v + 4w_b)}$$
$$b = \frac{2}{w_b(w_v + 4w_b)}$$

  w_b is the bin-width, w_v is the module-width, and d_x is the distance centers of modules v and the bin b in the x-direction. Figure 2.3(a) and Figure 2.3(b) show the original and the smoothed overlap functions, respectively. APlace (Kahng and Wang, 2005) and NTUplace3 (Chen *et al.*, 2008) use bell-shaped smoothing functions.

- **Helmholtz Smoothing:** Another approach is to approximate the smoothed density $\hat{D}_b(x, y)$ with Helmholtz equation.

$$\Delta \hat{D}_b(x, y) - \epsilon \hat{D}_b(x, y) = -D_b(x, y) \tag{2.14}$$

  where ϵ is a smoothing parameter, $\epsilon > 0$, and Δ is the Laplacian operator. mPL6 (Chan *et al.*, 2006) uses this approach with zero derivative boundary conditions.

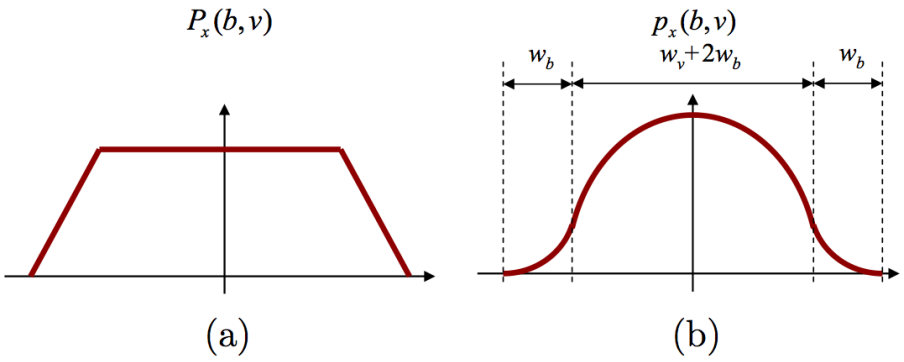

Figure 2.3: (a) The overlap function $P_x(b, v)$. (b) The smoothed overlap function $p_x(b, v)$.

- **Poisson Smoothing:** Another approach is to treat density as electrostatic potential and then use poisson equation 2.15 to approximate the smoothed density. FDP (Vorwerk *et al.*, 2004), Kraftwerk (Spindler *et al.*, 2008), and mFAR (Hu *et al.*, 2005) uses this approach.

$$\Delta \hat{D}_b(x, y) = -D_b(x, y) \tag{2.15}$$

2.5 Integration of Wirelength and Overlap

The wirelength models and overlap techniques discussed in the previous Sections 2.3 and 2.4 have to be unified so that it can used for global placement. Wirelength optimization tends to pull modules together and works contradictory to the overlap reduction, which pushes modules away from each other. In this section, we discuss various popular methods to integrate these two conflicting objectives.

2.5.1 Fixed Point Method

One most popular method is to feed the placement obtained from overlap reduction techniques back to the placement problem by adding fixed points and pseudo connections into the original netlist. Then the placement problem is again solved on the modified netlist. Figure 2.4 illustrates this approach. One fixed point is created at the target position obtained from overlap reduction of each module, and a pseudo connec-

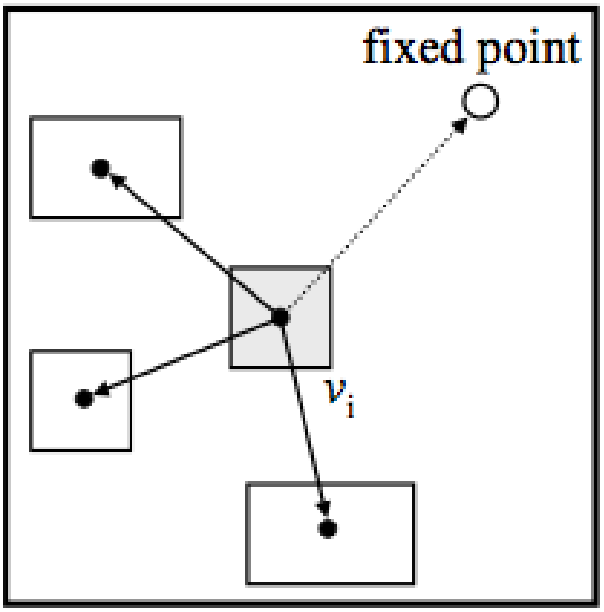

Figure 2.4: Fixed point method.

tion is made between them. This is adopted by placers like FDP (Vorwerk *et al.*, 2004), Kraftwerk2 (Spindler *et al.*, 2008) and mFAR(Hu *et al.*, 2005).

2.5.2 Penalty Method

Quadratic penalty method is used to solve a sequence of unconstrained minimization problems of the form

$$\min \quad W(x,y) + \lambda \sum_{b} (\hat{D}_b(x,y) - M_b)^2 \tag{2.16}$$

where λ is the weighting factor to balance the conflicting wirelength optimization and overlap reduction.

2.5.3 Partition Constraint

In partitioning-based overlap reduction discussed in Section 2.4.1, the cells are assigned to subregions instead of specific positions. Such assignment is still required to be linked back to the original placement problem for the later optimization. This can be achieved by fixing the center of gravity of modules assigned to the same subregion to the center

of the subregion.

2.6 Cost optimization

The popular types of the mathematical optimization used for cost minimization is described in the following subsections.

2.6.1 Quadratic Programming

The quadratic programming is one of the most common approaches to the placement problem. With the quadratic wirelength model, it can be solved as a quadratic optimization problem given by

$$\min_{x} \sum_{i,j} w_{x,ij}(x_i - x_j)^2 = \min_{x} \frac{1}{2}\mathbf{x^T Q_x x} + \mathbf{c_x^T x} + \mathbf{d_x} \tag{2.17}$$

where $w_{x,ij}$ represents the weight of the edge connecting modules i and j. The matrix $\mathbf{Q_x}$ is the Hessian which represents the hyperedge connectivity. If some modules are fixed, the Hessian is a symmetric, positive-definite matrix. The vector $\mathbf{c_x}$ represents connections between movable and fixed modules, and the vector $\mathbf{d_x}$ represents connections between fixed modules. This optimization problem is strictly convex and has a unique minima given by the solution of a single, positive-definite system of linear equations,

$$\mathbf{Q_x x} + \mathbf{c_x} = \mathbf{0} \tag{2.18}$$

Wirelength models like Bound2Bound can easily be integrated by changing the weights $w_{x,ij}$ accordingly. Overlap reduction techniques can be integrated using fixed point or partitioning by modifying the matrix $\mathbf{Q_x}$ or the vector $\mathbf{c_x}$ appropriately.

2.6.2 Nonlinear Programming

In general, nonlinear optimization problem for placement is solved using the penalty method of integration. Solving the nonlinear problem is usually very time consuming,

and hence the multilevel approach is often preferred. Placers like APlace (Kahng and Wang, 2005), mPL6 (Chan *et al.*, 2006) and NTUplace3 (Chen *et al.*, 2008) uses similar approaches.

2.7 Legalization

During legalization, all overlaps that exists after global placement are removed with minimal wirelength increase or module displacement. The relative order of the modules from global placement is maintained. One of the most popular methods for legalization is the Tetris-like greedy method (Hill, 2002). Modules are sorted according to their x coordinates, and then they are placed at the closet available positions with minimal cost. This algorithm is fast with negligible running time compared to that of the global placement. Another popular legalization method is called single-row placement. This method concerns about the optimal positions for the modules to be placed within the same row at one time, while their relative ordering is kept.

2.8 Detailed Placement

In the detailed placement stage, the standard cell positions is further optimized to improve the placement quality. The objective of detailed placement is to find a better position for each standard module in the available free spaces. Few popular approaches are described below.

- **Cell order polishing** permutes a small window of modules each time to find the best ordering by enumerating all possible orderings using the branch-and-bound method. The number of cells contained by the window is an important factor to control the tradeoff between the running time and solution quality.
- **Cell matching** is an efficient technique that can optimize more modules at the same time. The cell matching algorithm finds a group of exchangeable cells inside a given window, and formulates a bipartite matching problem by assigning the modules to available slots in the window. The assignment cost is the HPWL

difference of placing a module in different slots. The bipartite matching problem can be solved very quickly when the number of modules is smaller than 100. Compared with other detailed placement algorithms, cell matching can optimize the placement result more globally.

- **Global moving/**swapping moves each module to the optimal location among available whitespaces without changing the positions of other modules. This technique is especially useful when the design utilization is low. When design utilization is high, it may not be easy to find a whitespace to place the module. In this case, this technique tries to swap the module with a module within the optimal region to see if a better result can be obtained.

CHAPTER 3

Basics of Kraftwerk2

Kraftwerk2 (Spindler *et al.*, 2008) is one of the most popular academic placers at the time of writing this thesis. It is credited with achieved a good quality placement with less runtime. The algorithm is explained in detail in the following sections.

3.1 Initial Placement

In Kraftwerk2, initial placement is done by minimizing the quadratic cost function over few iterations using Equation 2.18. In each iteration, the Bound2Bound wirelength model is applied and the connection weights are recomputed. This is repeated over few iterations until the improvement in wirelength between iterations is less that 10%. At this point, the initial placement has minimal wirelength. But the modules are concentrated somewhere on the chip, usually at the center of the chip and there is significant overlap. After initial placement, the global placement routine is done, details of which are described in the next section.

3.2 Global Placement

During global placement, each iteration starts with determining the supply and demand system D using the Equation 3.1. The supply reflects the free space in the chip and the demand reflects the density of the modules.

$$D(x, y) = D^{dem}_{mod}(x, y) - D^{sup}_{mod}(x, y) \tag{3.1}$$

The overall demand is computed from individual demand from each module given by,

$$D^{dem}_{mod,i}(x, y) = d_{mod,i}.R(x, y; x_i - \frac{w_i}{2}, y_i - \frac{h_i}{2}, w_i, h_i) \tag{3.2}$$

where

$$R(x, y; x_{ll}, y_{ll}, w, h) = \begin{cases} 1 & \text{if } 0 \leq x - x_{ll} \leq w \wedge 0 \leq y - y_{ll} \leq h \\ 0 & \text{else.} \end{cases}$$

here, x_i and y_i are the x and y coordinates of the center of module i and w_i and h_i are the width and height of module i respectively. $x_l l$ and $y_l l$ is the coordinates of the lower left corner of the module.

Overall demand is the sum of individual module demands

$$D_{mod}^{dem} = \sum_{i=1}^{N} D_{mod,i}^{dem}(x, y) \tag{3.3}$$

Supply is given by,

$$D_{mod}^{sup}(x, y) = d_{sup}.R(x, y; x_{chip}, y_{chip}, w_{chip}, h_{chip}) \tag{3.4}$$

where x_{chip} and y_{chip} are lower left corner of the chip boundary and w_{chip} and h_{chip} are the width and the height of the chip respectively.

After the supply and demand system is computed on a discretized grid, the Bound2Bound wirelength is recomputed to calculate the new connection weights based on the current module positions. Since the objective function is convex, the minimum value can be found by solving Equation 2.18. In quadratic placement, each two-pin connection between modules can be viewed as an elastic spring. Hence the cost function represents the total energy of the spring system, and the derivative of an energy is a force. Hence minimizing the cost function is analogous to minimizing the overall energy of a mass spring system and the sum of all forces acting on each module is zero when every module is at its equilibrium position. Therefore, the wire force $\mathbf{F}^{net}$ between the modules is given by

$$\mathbf{F}^{net} = \mathbf{Q_x x} + \mathbf{c_x} \tag{3.5}$$

In Kraftwerk2, we use two additional forces, the hold force and the move force. The hold force is responsible for keeping the modules at its current position so that they dont collapse back to the minimal wirelength position while the quadratic cost is being

minimized. The hold force $\mathbf{F}^{hold}$ is hence the negative of the wire force and is given by,

$$\mathbf{F}^{hold} = \mathbf{Q_x x'} + \mathbf{c_x} \tag{3.6}$$

where $\mathbf{x'}$ is the current x coordinate of the modules. The move force is the force we apply on the modules to module them apart from each other and hence reduce overlap. The target fixed point $\dot{x}_i$ of each module i is given by

$$\dot{x}_i = x'_i - \frac{\partial}{\partial x}\hat{D}(x,y)\bigg|_{x_i,y_i} \tag{3.7}$$

where $\hat{D}(x,y)$ is the Poisson smoothed density function. Hence, the move force is defined as

$$\mathbf{F}^{move} = \dot{\mathbf{Q}}_\mathbf{x}(\mathbf{x} - \dot{\mathbf{x}}) \tag{3.8}$$

where $\dot{\mathbf{Q}}_\mathbf{x}$ is a diagonal matrix that contains the weights of the move force. Setting the sum of the wire force, the hold force, and the move force to zero, the following linear system can be obtained:

$$(\mathbf{Q_x} + \dot{\mathbf{Q}}_\mathbf{x})\mathbf{\Delta x} = -\dot{\mathbf{Q}}_\mathbf{x}\hat{\mathbf{D}}_\mathbf{x} \tag{3.9}$$

where $\mathbf{\Delta x} = \mathbf{x} - \mathbf{x'}$ The above equation is solved for both x and y directions separately. After solving, the module positions are updated and then a quality-control procedure is called to adjust the weights of the move force. The global placement is stopped if the *cell overlap* Ω is below a certain limit, e.g., below 20%. The *cell overlap* is defined as,

$$\Omega = 1 - \frac{\text{union of module areas}}{\text{sum of module areas}} \tag{3.10}$$

After global placement, the modules are legalized where remaining overlaps are removed and the modules are aligned to rows if necessary. Kraftwerk2 utilizes an approach similar to Tetris (Hill, 2002) to legalize standard cells. After legalization, a simple greedy detailed placement method is applied to improve the legal placement: Single modules are flipped, or pairs of neighboring modules are exchanged to improve the wirelength.

The following chapter explains our efforts to parallelize Kraftwerk2 and improve its performance in terms of runtime.

CHAPTER 4

Parallel Optimization of Kraftwerk2

Since the source code for Kraftwerk2 is not freely available, we developed our own implementation of the Kraftwerk2 algorithm. The entire codebase is in C++ and we have used Intel MKL (Intel, 2012) for computing the solution to the linear system of equations 2.18 and 3.9 and for solving the poisson equation needed to smooth the supply demand system. Since matrix $\mathbf{Q_x}$ is highly sparse and positive semi definite, we have used the CSR sparse matrix format to store it and we have used the Conjugate Gradient iterative method to solve it. Only the initial and global placement phases of Kraftwerk2 were implemented. Legalization and detailed placement was done using the NTUplace3 binary obtained from (NTUplace3, 2008).

The serial implementation was run on the ICCAD 2004 mixed-size benchmark suite (Adya *et al.*, 2004*b*) and the ISPD 2005 contest benchmark suite (ISPD, 2005). The final wirelength values we obtained are shown in Table 4.1.

As can be seen from Table 4.1, we were not able to reproduce the exact wirelength values that were claimed in (Spindler *et al.*, 2008). We believe this is due to different internal parameters, legalizer and detailed placement techniques used by us as compared to the authors of Kraftwerk2. On average, our implementation gave a wirelength that

Benchmark	Kraftwerk2	Own Implementation
ibm01	2.24	2.43
ibm04	7.63	8.20
ibm07	10.42	11.40
ibm10	30.15	34.50
ibm13	22.48	25.15
ibm16	54.17	58.13
ibm18	42.36	47.21
adaptec2	92.85	102.85
adaptec4	199.43	216.46
bigblue2	154.74	176.80
bigblue4	852.40	917.23

Table 4.1: Wirelength comparison between Kraftwerk2 and our implementation

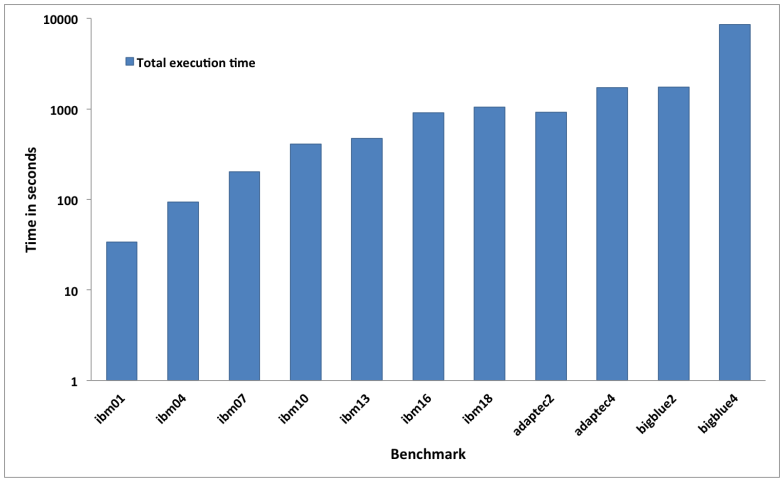

Figure 4.1: Total execution time for each benchmark (serial)

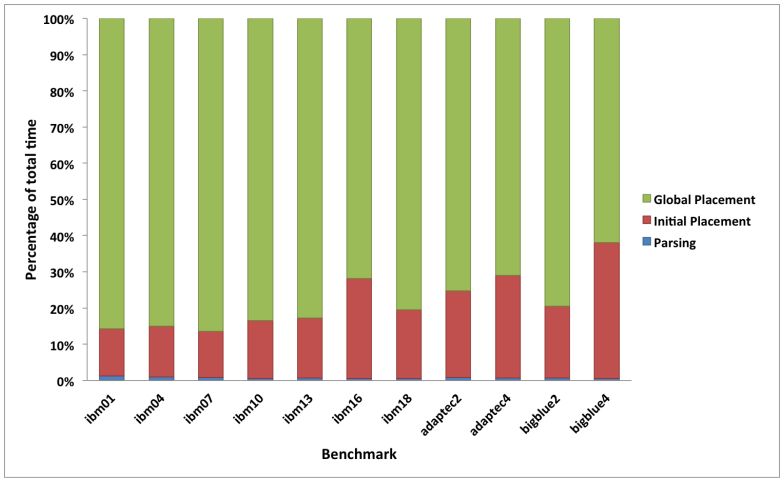

Figure 4.2: Percentage of total time spent in each phase

was around 10% more that the wirelength claimed in Spindler *et al.* (2008). The total execution time for each benchmark is shown in Figure 4.1.

4.1 Profiling

Before parallelization, we profiled our implementation to find the hot spots in our algorithm. The percentage of total time spent in each of the phases of the algorithm is shown in Figure 4.2.

As can be seen from Figure 4.2, the algorithm spends on an average 80% of its total execution time in the global placement phase. Figure 4.3 shows the profile of one iteration of global placement phase. We can see that a significant time taken by each

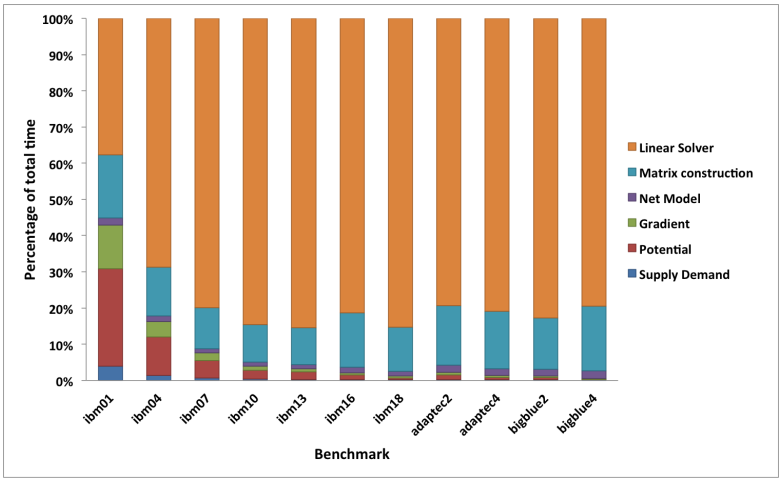

Figure 4.3: Time splitup of one global placement iteration

iteration is spent within the linear solver.

Hence our first effort was directed at improving the solver by parallelizing the solver on a multicore system, details of which are given in the next section.

4.2 Multicore CPU

The Conjugate Gradient solver uses a steepest descent method to iteratively converge to the optimal solution of the convex system being solved. The bottleneck of the solver is a Level-2 BLAS matrix-vector routine that has to be performed once every iteration within the solver. The Intel MKL Conjugate gradient solver is based on RCI (Reverse Communication Interface), which means that the solver can use a user-defined matrix-vector operation instead of the default one. We used a Sparse Level 2 routine which is parallelized internally to improve the performance. The speedup achieved by using parallelization is shown in Figure 4.4. The solver now gives almost **3x** speedup on using 12 parallel threads.

4.3 Preconditioning

An interesting feature of the iterative solver is that solver runtime is directly related to the number of iterations that is needed to the converge on the solution and this in-turn

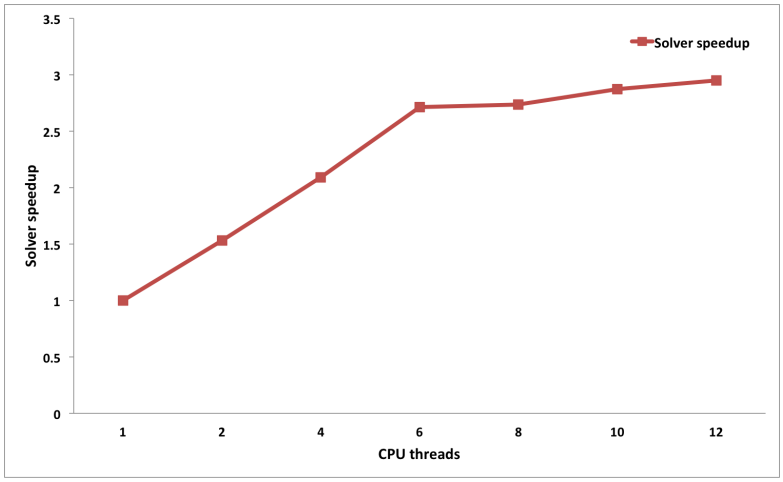

Figure 4.4: Solver speedup on using parallel BLAS

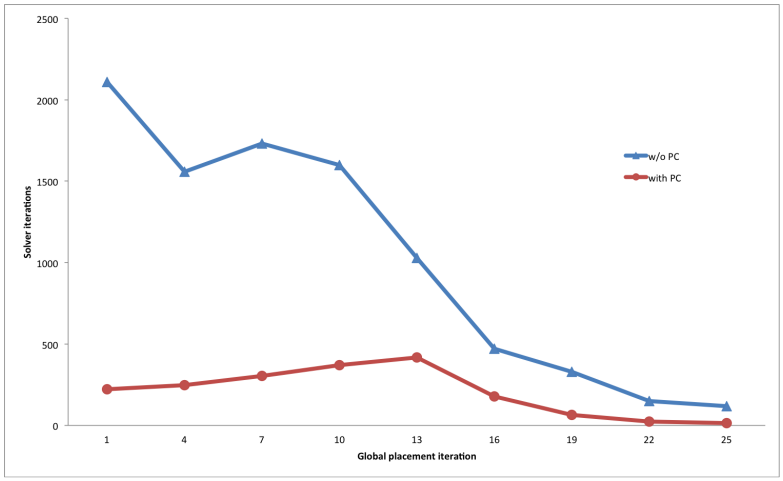

Figure 4.5: Internal solver iterations during global placement

is directly related to how far the solution is from the initial guess that is provided to the solver. In our implementation, we provide the placement from the previous global placement iteration as the starting guess for the solver. Hence, the average module movement per global placement iteration reflects how far the optimum position is from the initial guess (previous placement) for the given iteration. In Kraftwerk2, the average module movement per iteration has a general trend of increasing for the first few iterations and then decreasing for further iterations (Spindler *et al.*, 2008) till it converges. Our solver also follows this trend, which can be seen from Figure 4.5.

This solver iteration trend is undesirable and leads to high execution times for initial iterations. We have addressed this by using a preconditioner for the solver, which helps the solver to converge to the optimum faster. We have selected the Jacobi preconditioner composed of diagonal element of matrix $\mathbf{Q_x}$. The speedup achieved from using the

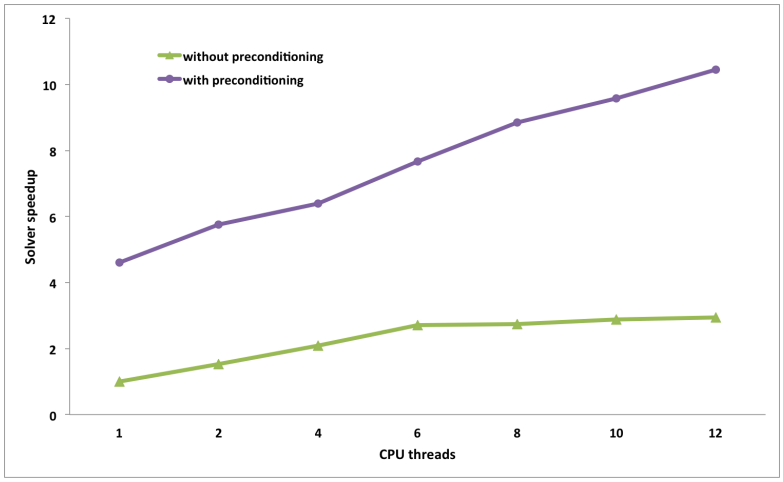

Figure 4.6: Solver speedup on using preconditioner

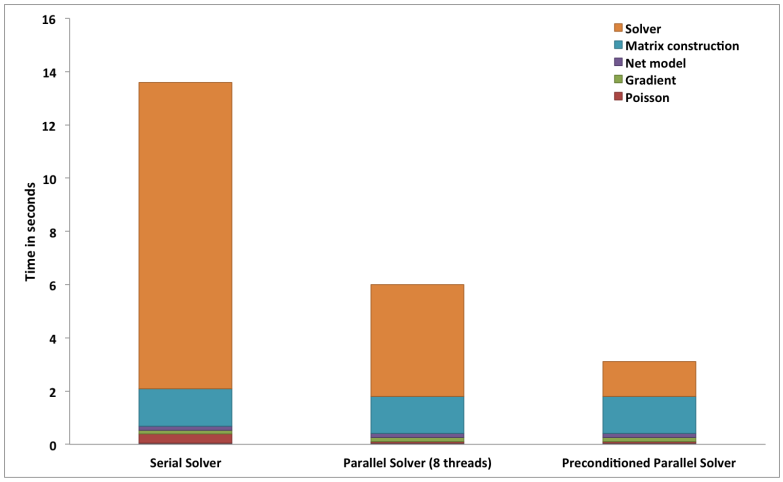

Figure 4.7: Improvement in global placement from using preconditioner

preconditioner can be seen from Figure 4.6. We get over **10x** solver speedup with 12 parallel threads and preconditioning compared to serial solver with no preconditioning.

The profile for the global placement iteration with parallel solver is shown in Figure 4.7. We can see that the time spent in the solver routine is now comparable with that of other routines like sparse matrix construction.

4.4 Wirelength model

Since we are using CSR format to represent the sparse matrix $\mathbf{Q_x}$ within our implementation, we need to completely rebuild the matrix in every iteration after the new Bound2Bound weights are calculated. Due to the limitations of the CSR format, it also

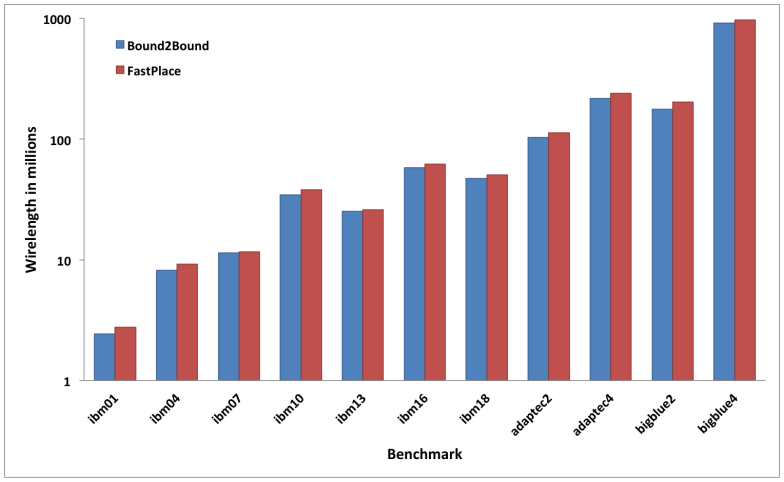

Figure 4.8: Change in solution quality with wirelength model.

not possible to get good results from parallelizing the matrix construction process with pthreads.

One way to work around this issue is by switching to other net model where the weights doesnt depend on the positions of the modules. We have used the net model adopted by FastPlace (Viswanathan and Chu, 2005). However, this leads to a loss in solution quality as the net model is no longer linearized. But we believe that the improvement in runtime justifies the loss in solution quality.

The loss in solution quality is shown in Figure 4.8. On average, we have a wirelength degradation of about 9%.

On switching to the FastPlace wirelength model, there is no longer a need to reconstruct the whole matrix in every iteration of global placement. The execution time comparisons between the net models are shown in Figure 4.9.

We can see that the solve is still the bottleneck of the global placement phase. As seen from Figure 4.4, we cannot get more than **3x** speedup from using more parallel threads in a CPU. BLAS routines perform very well in massively parallel systems like GPUs (Graphics Processing Units).

Our efforts to utilize the GPU in global placement are explained in the following section.

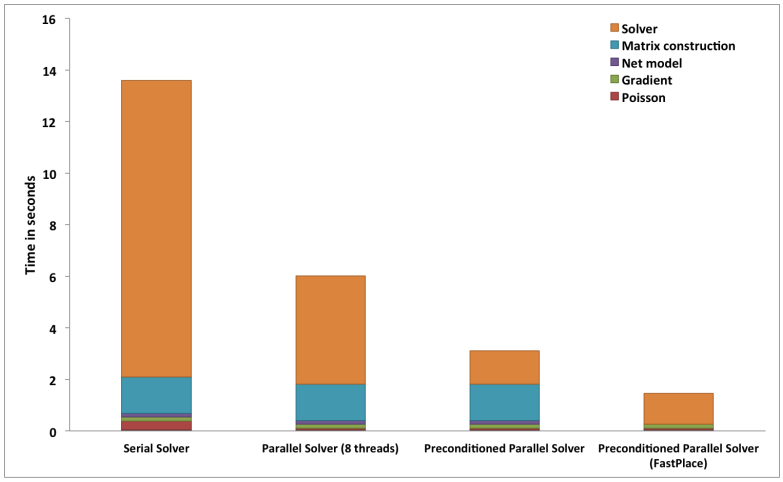

Figure 4.9: Improvement in global placement from using FastPlace net model

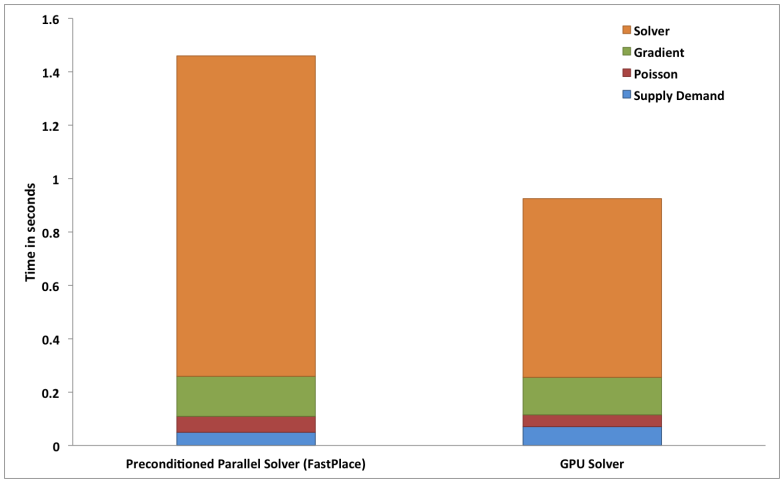

Figure 4.10: GPU Solver and CPU Solver comparison

4.5 GPU

We used the Preconditioned Conjugate Gradient Solver from CUSP Library (CUSP, 2012) to compute the solution to the linear equation on the GPU. CuPoisson (CuPoisson, 2012) was used to compute the poisson equation on the GPU. The improvement in global placement performance is shown in Figure 4.10.

Due to memory latency and memory bandwidth limitations, transferring data to the GPU and back is highly inefficient. Since we are using the FastPlace net model, we no longer have to compute the matrix during global placement. This opens up the possibility of performing the complete global placement routine on the GPU. Matrix $\mathbf{Q_x}$ and module locations are transferred to the GPU after initial placement and final position is transferred back to host memory (CPU memory) at the end of global placement.

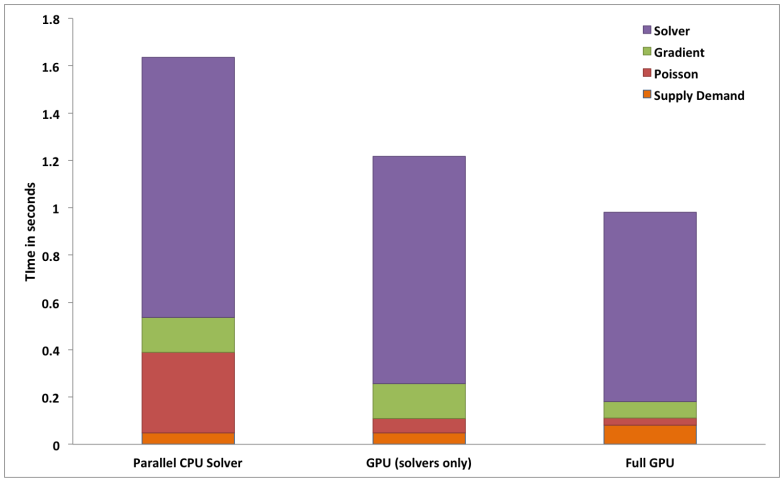

Figure 4.11: GPU Implementation comparison

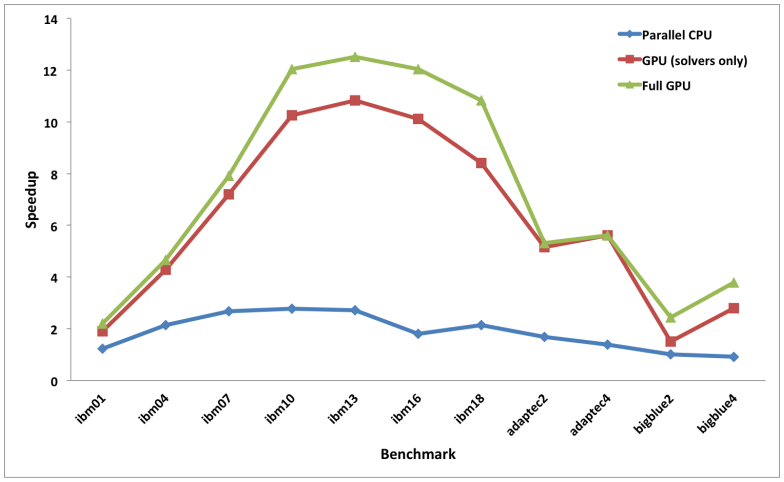

Figure 4.12: Speedup obtained on various benchmarks

The improvement in global placement iteration on performing entire global placement routine on GPU is shown in Figure 4.11.

The final speedup from all the various techniques we had explored is shown in Figure 4.12.

CHAPTER 5

Conclusion

There are two approaches to improve performance through parallelization. One method is to tweak the algorithm to introduce data independent computes and parallelization and the other method is to accelerate the computational bottlenecks of the algorithm using parallelization. We have explored the latter in the domain of analytical placement by adopting a well known placer, Kraftwerk2 and parallelizing the compute intensive areas of the algorithm without introducing major changes to the algorithm itself. We were able to get a maximum speedup of **3x** by utilizing parallel solvers on multicore CPU systems and a speedup of about **11x** on GPU systems. We were further able to achieve a maximum speedup to **13x** by performing the entire global placement routine on the GPU.

REFERENCES

1. **Adya, S.**, **S. Chaturvedi**, **J. Roy**, **D. Papa**, and **I. Markov**, Unification of partitioning, placement and floorplanning. *In Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*. 2004*a*. ISSN 1092-3152.

2. **Adya, S.**, **S. Chaturvedi**, **J. Roy**, **D. Papa**, and **I. Markov**, Unification of partitioning, placement and floorplanning. *In Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*. 2004*b*. ISSN 1092-3152.

3. **Agnihotri, A.** and **P. Madden**, Fast analytic placement using minimum cost flow. *In Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*. 2007.

4. **Brenner, U.** and **M. Struzyna**, Faster and better global placement by a new transportation algorithm. *In Design Automation Conference, 2005. Proceedings. 42nd*. 2005.

5. **Chan, T. F.**, **J. Cong**, **J. R. Shinnerl**, **K. Sze**, and **M. Xie**, mpl6: enhanced multilevel mixed-size placement. *In Proceedings of the 2006 international symposium on Physical design*, ISPD '06. ACM, New York, NY, USA, 2006. ISBN 1-59593-299-2. URL `http://doi.acm.org/10.1145/1123008.1123055`.

6. **Chen, T.-C.**, **T.-C. Hsu**, **Z.-W. Jiang**, and **Y.-W. Chang**, Ntuplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. *In Proceedings of the 2005 international symposium on Physical design*, ISPD '05. ACM, New York, NY, USA, 2005. ISBN 1-59593-021-3. URL `http://doi.acm.org/10.1145/1055137.1055188`.

7. **Chen, T.-C.**, **Z.-W. Jiang**, **T.-C. Hsu**, **H.-C. Chen**, and **Y.-W. Chang** (2008). Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **27**(7), 1228–1240. ISSN 0278-0070.

8. **CuPoisson** (2012). Cupoisson website.

9. **CUSP** (2012). Cusp website.

10. **Donath, W. E.**, Complexity theory and design automation. *In Proceedings of the 17th Design Automation Conference*, DAC '80. ACM, New York, NY, USA, 1980. ISBN 0-89791-020-6. URL `http://doi.acm.org/10.1145/800139.804563`.

11. **Hill, D.** (2002). Method and system for high speed detailed placement of cells within an integrated circuit design. US Patent 6,370,673.

12. **Hu, B.**, **Y. Zeng**, and **M. Marek-Sadowska**, mfar: fixed-points-addition-based vlsi placement algorithm. *In Proceedings of the 2005 international symposium on Physical design*, ISPD '05. ACM, New York, NY, USA, 2005. ISBN 1-59593-021-3. URL `http://doi.acm.org/10.1145/1055137.1055189`.

13. **Intel** (2012). Intel mkl website. URL http://software.intel.com/en-us/intel-mkl.

14. **ISPD** (2005). Ispd 2005 contest website.

15. **Kahng, A.** and **Q. Wang** (2005). Implementation and extensibility of an analytic placer. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **24**(5), 734–747. ISSN 0278-0070.

16. **Naylor, W. C.**, **R. Donelly**, and **L. Sha** (2001). Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer. US Patent 6,301,693.

17. **NTUplace3** (2008). Ntuplace3 binary.

18. **Ren, H.**, **D. Pan**, **C. Alpert**, **P. Villarrubia**, and **G.-J. Nam** (2007). Diffusion-based placement migration with application on legalization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **26**(12), 2158–2172. ISSN 0278-0070.

19. **Sechen, C.** and **A. Sangiovanni-Vincentelli** (1985). The timberwolf placement and routing package. *Solid-State Circuits, IEEE Journal of*, **20**(2), 510–522. ISSN 0018-9200.

20. **Sigl, G.**, **K. Doll**, and **F. M. Johannes**, Analytical placement: A linear or a quadratic objective function? *In Proceedings of the 28th ACM/IEEE Design Automation Conference*, DAC '91. ACM, New York, NY, USA, 1991. ISBN 0-89791-395-7. URL http://doi.acm.org/10.1145/127601.127707.

21. **Spindler, P.**, **U. Schlichtmann**, and **F. M. Johannes** (2008). Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, **27**(8), 1398–1411. ISSN 0278-0070. URL http://dx.doi.org/10.1109/TCAD.2008.925783.

22. **Taghavi, T.**, **X. Yang**, and **B.-K. choi**, Dragon2005: large-scale mixed-size placement tool. *In Proceedings of the 2005 international symposium on Physical design*, ISPD '05. ACM, New York, NY, USA, 2005. ISBN 1-59593-021-3. URL http://doi.acm.org/10.1145/1055137.1055191.

23. **Viswanathan, N.** and **C. Chu** (2005). Fastplace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **24**(5), 722–733. ISSN 0278-0070.

24. **Vorwerk, K.**, **A. Kennings**, and **A. Vannelli**, Engineering details of a stable force-directed placer. *In Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*. 2004. ISSN 1092-3152.