Parallelization of Iterative Heuristics for Performance Driven Low-Power VLSI Standard Cell Placement

Research Committee Project Seminar

Sadiq M. Sait

June 15, 2003

King Fahd University of Petroleum & Minerals
Dhahran
Introduction

Parallel CAD & Environments

Problem Formulation & Cost Functions

Fuzzification of Cost

Parallel Iterative Heuristics

Benchmarks & Experimental Setup

Tools

Objectives, Tasks

Conclude
Introduction & Motivation

- Problem: VLSI Cell-placement: NP-Hard
- Motivation for multi-objective VLSI optimization
  - Can include new issues: Power, Delay, Wirelength
- Motivation for using Non-deterministic Iterative Heuristics (Simulated Annealing, Genetic Algorithm, Simulated Evolution, Tabu Search, Stochastic Evolution)
  - Stable
  - Hill Climbing
  - Convergence
  - If engineered properly, then longer the CPU time (or more CPU power available) better the quality of solution (due to search space exploration)
Motivation for Parallel CAD

- Faster runtimes to achieve same results
- Larger problem sizes that have more CPU & memory requirements can be handled
- Better quality results
- Cost-effective technology compared to large multiprocessor supercomputer
- Utilization of abundant computer power of PCs that remain idle for a large fraction of time
- Availability and affordability of high speed transmission links, in the order of Gigabits/sec
Environments

- Consists of:
  - Programming Paradigm \(\rightarrow\) Software
  - Hardware Platform

- Need to choose:
  - A Hardware Platform, which is locally available
  - A Programming Paradigm, which is suitable for the selected platform
Four Possible Programming Paradigms:

- Explicit Message-Passing using Communication Libraries:
  - Parallel Virtual Machine (PVM) → C/C++/Fortran
  - Message Passing Interface (MPI) → C/C++/Fortran
- Data Parallel Languages
  - Message-Passing implemented through a parallel language
  - Compiler directives for specification of data distribution
    Example: High Performance Fortran (HPF)
- Shared-Memory Multiprocessing
  - Loop-level (fine-grained) parallelism
  - Identified through compiler directives
    Example: OpenMP standard
- Lightweight Threads (that use OS threads for parallelizing)
  - OS level threads used for hiding latencies (e.g., memory stalls)
    Examples: POSIX (Portable Operating System Interface) and Java threads
Our Choice:
- MPI and/or Multithreading with C/C++
- Explicit message-passing ➔ better control ➔ higher performance
- Multithreading useful to hide CPU stalls

Cluster of PCs (Cost effective)
- Affordable & abundantly available
- Freely available software tools

Our Choice:
- Cluster of Pentium III or IV PCs
- Connected through a GigE Switch
Cell Placement Problem:
- The process of placement consists of finding suitable locations for each cell on the entire layout.
- By suitable location we mean those that minimize a given objective, subject to certain constraints (width).

In this work we target the minimization of:
- Power
- Delay
- Wirelength

Constraint (width of the layout)
Cost function for Power

- In standard CMOS technology, power dissipation is a function of the clocking frequency, supply voltages and the capacitances in the circuit (recall that static power is zero except for due to leakage).

\[ P_{\text{total}} = \sum_{i \in V} p_i (C_i V_{dd}^2 f_{\text{clk}}) \beta \]

- where \( P_{\text{total}} \) is the total power dissipated
- \( p_i \) is the switching probability of gate \( i \)
- \( C_i \) represents the capacitive load of gate \( i \)
- \( f_{\text{clk}} \) is the clock frequency
- \( V_{dd} \) is the supply voltage and
- \( \beta \) is a technology dependent constant
The delay of any given long path is computed as the summation of the delays of the nets $v_1, v_2, ... v_k$ belonging to that path and the switching delay of the cells driving these nets. The delay of a given path is given by

$$T_p = \sum_{i=1}^{k} (CD_{vi} + ID_{vi})$$

where $CD_{vi}$ is the switching delay of the driving cell

$ID_{vi}$ is the interconnection delay that is given by the product of load-factor $LF_{vi}$ of the driving cell and the capacitance $C_{vi}$ of the interconnection net

$ID_{vi} = LF_{vi} \cdot C_{vi}$

We target only a selected set of long paths
Cost function for Wirelength

- Steiner tree approximation is used for each multi-pin net and the summation of all Steiner trees is considered as the estimated wire-length of the proposed solution

\[ X = \frac{\sum_{i=1}^{n} x_i}{n} \quad Y = \frac{\sum_{i=1}^{n} y_i}{n} \]

- where \( n \) is the number of cells contributing to the net

\[ SteinerTree = B + \sum_{j=1}^{k} D_j \]

- where \( B \) is the length of the bisecting line,
- \( k \) is the number of cells contributing to the net and
- \( D_j \) is the perpendicular distance from cell \( j \) to the bisecting line
Cost

- The cost of the width is given by the maximum of all the row widths in the layout.
- Layout width is constrained not to exceed a certain positive ratio $\tilde{U}$ to the average row width $w_{avg}$.
- This can be expressed as

$$Width \leq (1 + \alpha).w_{avg}$$
Three objectives are optimized simultaneously, with width constraint. This is done using fuzzy logic to integrate multiple objectives into a scalar cost function.

This is translated as OWA (ordered weighted average) fuzzy operator, and the membership \( \mu(x) \) of a solution \( x \) is a fuzzy set given by:

\[
\mu(x) = \begin{cases} 
1 & \text{if } x \text{ satisfies all objectives} \\
\frac{x}{W} & \text{if } x \text{ satisfies less than all objectives} \\
0 & \text{otherwise}
\end{cases}
\]
For combinatorial optimizations, three types of parallelization strategies are reported in literature:

- The operation within an iteration of the solution method are parallelized (for example, cost computation in Tabu search).
- The search space (problem domain) is decomposed.
- Multi-search threads (for example, built-in library in PVM, generates threads and distributes to different processors).
Crainic et. al. classify Parallel Tabu Search heuristic based on a taxonomy along three dimensions

- Control Cardinality: 1-control, p-control (1-control is one master in master slave; in p-control every processor has some control over the execution of the algorithm such as maintaining its own Tabu list, etc.)

- Control and Communication type:
  - Rigid Synchronization (RS, slave just executes masters instruction and synchronizes)
  - Knowledge Synchronization (KS, more information is sent from the master to the slave)
  - Collegial (C, asynchronous type, power to execute is given to all processors)
  - Knowledge Collegial (KC, similar to above, but master gives more information to the slaves)
Parallelization of Tabu Search

- Search Differentiation
  - SPSS (single point single strategy)
  - SPDS (single point different strategy)
  - MPSS (multiple point single strategy)
  - MPDS (multiple point multiple strategy)

- Parallel Tabu Search can also be classified as:
  - Synchronous
    - 1-control: RS, or 1-control: KS
  - Asynchronous
    - p-control: C, or p-control: KC
Related work (Parallel TS)

- Yamani et. al present Heterogeneous Parallel Tabu Search for VLSI cell placement using PVM
- They parallelized the algorithm on two levels simultaneously;
  - On the higher level, the Master starts a number of Tabu Search Workers (TSW) & provides them with a same initial solution
  - On the lower level, Candidate list is constructed, where each TSW starts a Candidate List Worker (CLW)
- Based on the taxonomy aforementioned, the algorithm can be classified as
  - Higher level: p-control, Rigid Sync, MPSS
  - Lower level: 1-control, Rigid Sync, MPSS
Most of the GA parallelization techniques exploit the fact that GAs work with population of chromosomes (solutions).

- The population is partitioned into sub-populations which evolve independently using sequential GA.
- Interaction among smaller communities is occasionally allowed.
- The parallel execution model is a more realistic simulation of natural evolution.
- The reported parallelization strategies are:
  - Island model
  - Stepping stone model
  - Neighborhood model
Parallelization strategies for GA

- Island model
  - Each processor runs sequential GA
  - Periodically subsets of elements are migrated
  - Migration is allowed between all subpopulations

- Stepping stone model
  - Similar to Island model but communication is restricted to neighboring processor only
  - This model defines fitness of individuals relative to other individuals in local subpopulation

- Neighborhood model
  - In this model, every individual has its own neighborhood, defined by some diameter
  - Suitable for massively parallel machines, where each processor is assigned one element
Related work (Parallel GA)

- Mohan & Mazumder report VLSI standard cell placement using a network of workstations.
- Programs were written in C, using UNIX's rexec (remote execution) and socket functions.
- They summarized their findings as:
  - The result quality obtained by serial and parallel algorithms were same.
  - Statistical observations showed that for any desired final result, the parallel version could provide close to linear speed-up.
  - Parameter settings for migration and crossover were identified for optimal communication pattern.
  - Static and dynamic load balancing schemes were implemented to cope with network heterogeneity.
ISCAS-89 benchmark circuits are used

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No. of gates</th>
<th>No. of paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>S298</td>
<td>136</td>
<td>150</td>
</tr>
<tr>
<td>S386</td>
<td>172</td>
<td>205</td>
</tr>
<tr>
<td>S641</td>
<td>433</td>
<td>687</td>
</tr>
<tr>
<td>S832</td>
<td>310</td>
<td>240</td>
</tr>
<tr>
<td>S953</td>
<td>440</td>
<td>583</td>
</tr>
<tr>
<td>S1196</td>
<td>561</td>
<td>600</td>
</tr>
<tr>
<td>S1238</td>
<td>540</td>
<td>661</td>
</tr>
<tr>
<td>S1488</td>
<td>667</td>
<td>557</td>
</tr>
<tr>
<td>S1494</td>
<td>661</td>
<td>558</td>
</tr>
<tr>
<td>C3540</td>
<td>1753</td>
<td>668</td>
</tr>
<tr>
<td>S9234</td>
<td>5844</td>
<td>512</td>
</tr>
<tr>
<td>S13207</td>
<td>8744</td>
<td>708</td>
</tr>
<tr>
<td>S15850</td>
<td>10470</td>
<td>512</td>
</tr>
</tbody>
</table>
Experimental Setup

- Consists of
  - Hardware:
    - Cluster of 8 machines, x86 architecture, Pentium 4, 2GHz clock speed, 256 MB of memory
    - Cluster of 8 machines, x86, Pentium III, 600 MHz clock speed, 64 MB memory
  - Connectivity:
    - Gigabit Ethernet switch
    - 100 MBit/Sec Ethernet switch
  - OS: Linux
  - Available Parallel Environments: PVM, MPI
Profiling tools

- Intel® VTune Performance Analyzer
- Gprof (built in Unix tool)
- VAMPIR (an MPI profiling tool)
- upshot (built in tool with MPIch implementation)

Debugging tools

- Total View (from Etnus)
- GNU debugger (gdb)
The proposed research work will build upon our previous attempts that focused on design and engineering sequential iterative algorithms for VLSI standard cell placement.

The primary objective of the present work is to accelerate the exploration of search space by employing the available computational resources for the same problem and with the same test cases.
Outline

1. Setting up the cluster environment
2. Porting of code, followed by review and analysis of sequential implementation to identify performance bottlenecks
3. Literature review of the work related to parallelization of iterative heuristics
4. Investigation of different acceleration strategies
5. Collection of data and tools for implementation, analysis and performance evaluation
6. Design and implementation of parallel heuristics
7. Compilation of results and comparison
8. Documentation of the developed software
Current Status

- Two clusters ready, switches procured, initial experiments indicate that existing infrastructure adequate for starting of project
- All serial code initially developed on PCs/Windows now being ported to the cluster environment running Linux
- Insight into all code is being obtained, and bottlenecks being identified using profiling tools
- Initial experiments conducted with Tabu search show promising results. There is a lot to be done.
- Some experiments are being carried out using Genetic Algorithms.
Trivial Implementation

- The first synchronous parallel tabu search implementation procedure is built according to 1-control, Rigid Synchronization (RS), SPSS strategy.
- In this implementation, the master process executes the tabu search algorithm, while the candidate list (neighborhood size N), is divided among p slaves.
- Each of the slaves evaluates the cost of the \((N/p)\) moves (swapping the cells), and report the best of them to the master.
- The master collects the solutions from all slaves and accepts the best among them to be the next initial solution, for the next iteration.
- There is communication and exchange of information between the master and slaves after every iteration.
Test circuits (c3540 & s9234)
Observations of the 1-control, RS, SPSS strategy

- The total runtime decreased as the number of processors increased, in achieving the same quality of solution (except for c3540 circuit, for 2 processors, where communication was more then computation)
- The speed-up in terms of time, for the large circuit (s9234) was
  - 1.42 for 1 slave processor
  - 1.82 for 2 slave processors
  - 2.72 for 3 slave processors
  - 3.50 for 4 slave processors
  - 4.46 for 5 slave processors
value of the Project

- It is expected that our findings will help in addressing other NP-hard engineering problems
- The results can be used by other investigators in academia and industry to enhance existing methods
- A lab environment setup will provide support for future research, and enable utilization of existing idle CPU resources. The environment will also help train graduate students in VLSI physical design research, iterative algorithms, and parallelization aspects.
The project aims at accelerating runtime of iterative non-deterministic heuristics by exploiting available unused CPU resources. It is expected that insight on how the implementation of heuristics behaves in a parallel environment will lead to newer parallelization strategies. Initial experiments indicate that the available environment can provide sufficient support for such type of work.
Thank you
Hardware

Choices for Parallel Computing Hardware:

- Shared-Memory Multiprocessors
  - Too expensive
  - Require specialized software tools
  - None available locally

- Cluster of PCs
  - Affordable & abundantly available
  - Freely available software tools

Our Choice:

- Cluster of Pentium III or IV PCs
- Connected through a GigE Switch