Hindawi
Mobile Information Systems
Volume 2018, Article ID 2921451, 8 pages
https://doi.org/10.1155/2018/2921451

Research Article

HotSpot Thermal Floorplan Solver Using Conjugate Gradient to Speed Up

Zhonghua Jiang and Ning Xu

School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China

Correspondence should be addressed to Zhonghua Jiang; jzh800@126.com

Received 9 November 2017; Revised 5 February 2018; Accepted 20 February 2018; Published 5 April 2018

Academic Editor: Yuh-Shyan Chen

Copyright © 2018 Zhonghua Jiang and Ning Xu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We proposed to use the conjugate gradient method to effectively solve the thermal resistance model in HotSpot thermal floorplan tool. The iterative conjugate gradient solver is suitable for traditional sparse matrix linear systems. We also defined the relative sparse matrix in the iterative thermal floorplan of Simulated Annealing framework algorithm, and the iterative method of relative sparse matrix could be applied to other iterative framework algorithms. The experimental results show that the running time of our incremental iterative conjugate gradient solver is speeded up approximately 11x compared with the LU decomposition method for case ami49, and the experiment ratio curve shows that our iterative conjugate gradient solver accelerated more with increasing number of modules.

1. Introduction

With the constant improvement of the chip speed, power and temperature of a chip also increase accordingly. To cope with the increasing temperature of chips, the thermal aware floorplan method is widely used to avoid hotspots on chips in physical design. It makes the thermal problem to be more critical for physical design quality. In [1, 2], the authors employ the hierarchical thermal model to decrease the modules' maximum temperature in chip physical design. They use B^*-tree [3, 4] to represent the floorplan/placement with Simulated Annealing optimization algorithm. The authors' hierarchical thermal floorplan/placement includes two critical steps. First, they cluster the modules by power density, and then they use the Gauss–Seidel method to solve the thermal linear system. In the [1, 2], the authors only have done a theoretical analysis between the Gauss–Seidel method and the traditional LU decomposition matrix solver by loop iterative times. They assume that the time complexity of the traditional LU decomposition matrix solver is $2n^3/3$ and compare the Gauss–Seidel method real loop iterative times with cn^2 instead of the real program or solver CPU running time.

In [5], the authors compare the program running time; this comparison is not obvious because the program running time includes other cost computations and SA's iterative time. In order to better compare the linear solver, in this study, we improve the timing statistical method and join the other precondition methods, such as the SSOR precondition.

In [8], exponentially increasing power densities in current day designs due to aggressive technology scaling has resulted in temperature being one of the primary design constraints along with others like timing, area, and power. A lot of design techniques are being adopted during the physical design stage to minimize the power, apart from the architectural techniques like throttling for dynamic thermal management. In [8], the authors propose a practical methodology for better thermal management by floorplan modifications based on thermal hotspots obtained through dynamic simulations, without disturbing the logical connectivity information. This methodology definitely warrants the benefits which can be readily realized by doing this analysis early in the design cycle. This can also improve the placement of the thermal sensors and boost additional performance which can be extracted by their delayed triggering, considering the lateral spreading due to better floorplanning.

In [9], with the continuing scaling of CMOS technology, on-chip temperature and thermal-induced variations have become a major design concern. To effectively limit the high temperature in a chip equipped with a cost-effective cooling system, thermal-specific approaches, besides low-power techniques, are necessary at the chip design level. The high temperature in hotspots and large thermal gradients are caused by the high local power density and the nonuniform power dissipation across the chip. With the objective of reducing power density in hotspots, the authors proposed two placement techniques that spread cells in hotspots over a larger area. Increasing the area occupied by the hotspot directly reduces its power density, leading to a reduction in peak temperature and thermal gradient. To minimize the introduced overhead in delay and dynamic power, they maintained the relative positions of the coupling cells in the new layout. They compared the proposed methods in terms of temperature reduction, timing, and area overhead to the baseline method, which enlarges the circuit area uniformly. The experimental results showed that our methods achieve a larger reduction in both peak temperature and thermal gradient than the baseline method. The baseline method, although reducing peak temperature in most cases, has little impact on thermal gradient.

In [10], with the thermal effect, improper analog placements may degrade circuit performance because the thermal gradient can affect electrical characteristics of the thermally sensitive devices. To mitigate the thermal effect in analog layout design, it is required to reduce thermally induced mismatches among matched devices in addition to eliminating thermal hot spots. The study presented major challenges arising from the chip thermal gradient for analog placement , introduced nonuniform and uniform thermal profiles as well as the corresponding placement configurations, surveyed key existing techniques for analog placement under nonuniform and uniform thermal profiles, and provided the experimental results for analog placement with thermal consideration.

In [11], the work developed a thermal aware placer, ThermPL, to abate both on-chip peak temperature and thermal gradient by developing thermal force and padding techniques cooperated with rough legalization in the force-directed global placement. Thermal padding is firstly adopted to reduce local power density. To make use of thermal force, the authors used the thermal gain basis to fast and accurately capture the temperature distribution of a placement and effectively calculate the thermal contribution of cells based on the thermal locality. Then, they utilized the proposed innate thermal force assessed through thermal criticality and capabilities to spread cells away from hotspots. With the thermal gain basis, ThermPL can efficiently obtain the thermal profile of placement with the maximum error of 0.65% compared with a commercial tool. Experimental results show that ThermPL can provide 7% and 19% reduction on average in peak temperature and thermal gradient, respectively, within only 4.6% wirelength overhead.

In [12], with the thermal effect, improper analog placements may degrade circuit performance because the thermal impact from power devices can affect electrical characteristics of the thermally sensitive devices. There is not much previous work that considers the desired placement configuration between power and thermally sensitive devices for a better thermal profile to reduce the thermally induced mismatches. The study first introduced the properties of a desired thermal profile for better thermal matching of the matched devices. It then presented a thermal-driven analog placement methodology to achieve the desired thermal profile and to consider the best device matching under the thermal profile while satisfying the symmetry and the common-centroid constraints. Experimental results based on real analog circuits show that the proposed approach can achieve the best analog circuit performance/accuracy with the least impact due to the thermal gradient, among existing works.

In this study, we embed the iterative conjugate gradient method into HotSpot floorplan to compare the real CPU running time different solvers with the same compiler and running environment.

The conjugate gradient solver was imported into the thermal floorplan tool HotSpot [13], comparing with its previous LU decomposition solver. Then, we can compare the running time with different solvers of thermal floorplan between conjugate gradient method and LU decomposition solver. The thermal floorplan solver is switched by the program command line parameter with the same running environment such as CPU and memory, GCC version, and compiler's option. It is more convective to compare CPU time with two solvers than theoretical analysis about loop iterative times. We also use the SSOR and Jacobi preconditions to accelerate the conjugate gradient solver.

HotSpot thermal aware floorplan employs the thermal model to compute the blocks' temperature; the thermal temperature metric is combining with other area and wire length metrics, and it is a relative sparse matrix in the HotSpot thermal model of iterative SA framework algorithm. The HotSpot thermal floorplan can decrease the maximum of block temperature by evenly distributing the power density, avoiding hotspots in the floorplan step of physical design. We import an iterative method to solve this kind of relative sparse linear system in the thermal model of floorplan. This paper's contributions include the following:

(i) The relative sparse matrix is defined. It can speed up linear system solver convergence by the iterative sparse method.

(ii) The conjugate gradient iterative method is imported in the HotSpot floorplan thermal model. It is an efficient algorithm that can reduce the running time by accelerating the linear solver in hotspot.

This paper is organized as follows. Section 2 introduces the HotSpot floorplan flow. Relative sparse matrix definition and the thermal resistance model are given in Section 3 and Section 4, respectively. Section 5 shows the result of experiments, and conclusions and future research are given in Section 6.

2. HotSpot Thermal Floorplan

The VLSI physical design floorplan is to place the blocks without overlap in the silicon die, and the floorplan algorithm needs to obey the chip constraint, optimizing area, wire

length, and thermal temperature metrics' cost. The placement temperature is solved by linear equations in the thermal model.

2.1. Introduction of Hot Thermal Floorplan. The floorplan/placement of physical design is a critical step for the thermal aware design. Hot floorplan is a thermal aware tool to decrease the module temperature and avoid the hotspot convergence. The hot floorplan merges the thermal cost with the traditional area and wire length cost into iterative Simulated Annealing algorithm, and it is time-consuming to solve the module temperature in the iterative Simulated Annealing algorithm.

We also use the HotSpot model to guide thermal floorplan/placement to do static temperature computer and modules' temperature statistics, and then the thermal cost is integrated with the other area and wire length to do thermal aware physical design.

HotSpot builds a thermal model to compute the dynamic block or grid temperature on chip. We import the conjugate gradient solver to accelerate the thermal block model computation in floorplan SA algorithm.

3. Relative Sparse Matrix of Iterative Framework Algorithm

There are more sophisticated algorithms to solve sparse linear equations, avoiding to process zero entries of sparse matrix [6, 7], such as the iterative method of linear systems. The relative sparse matrix is not a strict sparse matrix and it tends to be a dense matrix; relative sparse matrix means that there is a few "interesting" entry between one matrix and another.

3.1. Relative Sparse Matrix. Relative sparse matrix definition: if the matrix $R = A_{k+1} - A_k$ is sparse, there is a few nonzero values of matrix R items. We define that matrices A_k and A_{k+1} are relatively sparse; in other words, the matrix A_{k+1} is a sparse matrix relative to matrix A_k. Assume there are two linear equations $A_k * x = b$ and $A_{k+1} * x = b$, the order of solving linear equations is as follows: the first step is solving $A_k * x = b$, the second step is solving $A_{k+1} * x = b$ in sequentially iterative framework algorithm; for example, Simulated Annealing algorithm solves $A_k * x = b$ and $A_{k+1} * x = b$ sequentially with the same vector b, in a linear system with different matrices from A_k to A_{k+1}.

In this case of sequentially relative sparse matrix, the iterative methods are employed to solve the relative sparse linear equations to reduce the number of iterations for the convergence. The detail operations are described as follows:

In the first step, the linear system $A_0 * x = b$ is solved trivially, obtaining the solution x_0, and then we reuse the solution x_0 as the initial estimate value for the linear equations $A_1 * x = b$. In the same way, we reuse the solution x_k of $A_k * x = b$ to be initial estimate value of $A_{k+1} * x = b$, $k \geq 0$ sequentially.

In this case of the sequence relative sparse matrix, if we set the initial estimate value of $A_{k+1} * x = b$ equal to x_k, x_k is the previous solution of $A_k * x = b$, and it can speed up the iterative method convergence. Because the changes of the matrix from A_k to A_{k+1} is little, even though the matrices A_k and A_{k+1} are full matrix not traditional sparse matrix about entry densities, the matrix A_{k+1} is a sparse matrix relative to the matrix A_k.

We can call this relative sparse linear system computation as the incremental updating solution method.

In the iterative algorithm, the previous iterative solution preserved as intermediate solution is the initial estimate value for current iteration in the linear system.

4. Relative Sparse Matrix in Thermal Floorplan of SA Framework Algorithm

4.1. Thermal Model Introduction. In the floorplan of VLSI physical design, the circuit modules are randomly placed in the die using Simulated Annealing; once the Simulated Annealing generates a floorplan of circuit modules, we calculate the cost metric of die area, wire length between circuit modules, and the maximum temperature of circuit modules by the thermal model.

The Simulated Annealing is an iterative optimizing algorithm; the thermal model is incorporated into the SA (Simulated Annealing). The thermal conduct in the die is complex [14], and it can be an abstracted thermal resistant model [15]:

$$R * T = P \Rightarrow T = R^{-1} * P, \tag{1}$$

where T and P are the vectors representing temperature and power consumption, respectively; thermal resistance R is the square matrix and symmetric matrix. Once the circuit blocks are determined, the blocks' power P vector will not change, and it is a constant vector. Thermal resistance matrix R will change entry values according to the placement detail, and it is a dense matrix instead of the sparse matrix, but it matches the relative sparse matrix definition in iterative Simulated Annealing framework algorithm.

The Simulated Annealing algorithm changes placement, a few from one stage to another, computing new cost of metrics, for example, moving a block from one location to another unused location, and this perturbation will only change one block's location in placement so that the thermal conduct between blocks and most block's temperature changes little too. Here the updating block's thermal resistance matrix R is a dense matrix but has a few "interesting" changes about thermal resistance matrix items, and it is a relative sparse matrix between R_{n+1} and R_n. The new thermal resistance matrix R_{n+1} is a relative sparse matrix with R_n.

4.2. LU Decomposition Solving Linear Equation. Solving linear equations gave a system of linear equations in the matrix form:

$$Ax = b. \tag{2}$$

Given matrix A and vector b, the solution x is needed to be solved. The matrix A is LUP decomposed such that $PA = LU$. The linear equations could be transformed into LU form equivalently as

$$\mathrm{LU}x = Pb. \tag{3}$$

The LU solver is done in two logical steps:

(i) First step: solving the lower triangular matrix linear equations, $\mathrm{L}y = Pb$ for y

(ii) Second step: solving the upper triangular matrix linear equations, $Ux = y$

The cost of solving a system of linear equations is approximately $2n^3/3$ floating point operations if the matrix A has size n [16].

The LU decomposition is the direct solver method of the linear equation.

4.3. Incrementally Iterative Conjugate Gradient Solver and Convergence. There are many iterative linear solver methods including conjugate gradient, Gauss–Seidel, and successive over relaxation. In this study, the conjugate gradient method is used to solve the thermal model in floorplan.

4.3.1. Convergence of Incrementally Iterative Conjugate Gradient Solver. The conjugate gradient method is convergence if the matrix is symmetric and positive definite. The condition number associated with the linear equation $Ax = b$ gives a bound on how inaccurate the solution x will be after approximation. The condition number of matrix is the product of the two operator norms:

$$\kappa(A) = \left\|A^{-1}\right\| \cdot \|A\|. \tag{4}$$

If A is normal, then $\kappa(A) = |\lambda_{\max}(A)/\lambda_{\min}(A)|$, where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are maximum and minimum values of eigenvalues of a matrix A, respectively. The convergence of CG depends on the condition number of matrix $\kappa(A)$ which is equal to $|\lambda_{\max}(A)/\lambda_{\min}(A)|$.

Denoting initial guess for x_0, at starting of the SA algorithm, we can assume that $x_0 = [0, \ldots, 0]$; if we get x after the first time linear solver, then the conjugate gradient solver will reuse the previous x_k as the next time initial estimate value x_{k+1}, $k \geq 0$. This incremental updating solution method can accelerate the solver convergence.

The conjugate gradient method inspires $Ax = b$ solution, and x_* is also unique in minimizing the following quadratic function:

$$f(x) = \frac{1}{2}x^T Ax - x^T b, \quad x \in R^n. \tag{5}$$

This suggests taking the first basis vector p_1 to be the negative of the gradient of $f(x)$ at $x = x_0$, the gradient of $f(x)$ equals $Ax_0 - b$, and we take $p_1 = b - Ax_0$.

It is conjugate to gradient between the vectors.

Let r_k be the residual at the kth step:

$$r_k = b - Ax_k. \tag{6}$$

Note that r_k is the negative gradient of f at $x = x_k$, so the gradient descent method would be to move in the direction r_k. Here, we insist that the directions p_k be conjugate to each other. We also require that the next search direction be built out of the current residue and all previous search directions, which is reasonably enough in practice.

4.3.2. Pseudocode of Conjugate Gradient Algorithm. The algorithm is detailed below for solving $Ax = b$, where A is a real, symmetric, positive-definite matrix. The input vector x_0 can be an approximate initial solution or 0.

The pseudocode of conjugate gradient solver is shown in Algorithm 1.

4.3.3. Precondition of Conjugate Gradient Method. In most cases, preconditioning is necessary to ensure fast convergence of the conjugate gradient method. The preconditioned conjugate gradient method takes the following form.

We consider a preconditioned system of

$$M^{-1}Ax = M^{-1}b, \tag{7}$$

where M is a nonsingular matrix.

Jacobi preconditioning: the simplest preconditioner consists of just the diagonal of the matrix:

$$m_{i,j} = \begin{cases} a_{i,j} & \text{if } i == j \\ 0 & \text{else } i! = j. \end{cases} \tag{8}$$

This is known as the Jacobi preconditioner.

The SSOR preconditioner, like the Jacobi preconditioner, can be derived from the coefficient matrix without any work. If the original, symmetric matrix is decomposed as

$$A = D + L + L^T, \tag{9}$$

in its diagonal, lower, and upper triangular part, the SSOR matrix is defined as

$$M = (D + L)D^{-1}(D + L)^T. \tag{10}$$

The pseudocode of the conjugate gradient solver with preconditioned is shown as Algorithm 1.

4.3.4. Incrementally Inherits Initializing Estimate Value from Previous Solution. The HotSpot thermal floorplan is using the iterative Simulated Annealing algorithm. The Simulated Annealing algorithm changes the placement from one to another, only one or two blocks' location, and the most blocks' temperatures change a little. If the initialize estimate value inherits from the previous temperature result, it can reduce the number of iteration times for convergence. It is the reason that the SA framework floorplan algorithm employs the iterative conjugate gradient solver to accelerate the convergence for the thermal model.

The pseudocode of SA thermal floorplan algorithm with the conjugate gradient thermal solver is shown in Algorithm 2.

4.4. Stopping Criteria for Iteration Solver. The residual r_{norm} is computed as follows:

$$r_{\mathrm{norm}} = (r, r) = ((b - A * x), (b - A*x)). \tag{11}$$

Require: $f(x)$: objective function; $x0$: initial solution;
Ensure: optimal x^*
(1) int Iter = 0;
(2) double α, β, rp, rpold, b_{norm}, r_{norm};
(3) vector r, p, q, z;
(4) initial $r = b \ ? \ A \ ? \ x$;
(5) $r_{\text{norm}} = (r,r)$;
(6) **while** (r_{norm} < precision) **do**
(7) **if** using preconditioned then
(8) $z = M^{-1}r$; (using difference preconditioned)
(9) **else**
(10) $z = r$; (no preconditioned)
(11) **end if**
(12) rp = (r,z);
(13) **if** Iter == 1 **then**
(14) $p_0 = z$
(15) **else**
(16) β = rp/rpold;
(17) $p = z + \beta * p$
(18) **end if**
(19) $q = A * p$;
(20) $\alpha = (r,z)/(p,q)$;
(21) $x = x + \alpha * p$;
(22) $r = r - \alpha * q$;
(23) $r_{\text{norm}} = (r,r)$;
(24) rpold = rp;
(25) **end while**

Algorithm 1: The conjugate gradient algorithm as an iterative method.

If and only if $r_{\text{norm}} < \text{aim}_{\text{precision}}$, the iterations are terminated after the residual is less small than the desired precision.

The overall iterative times of the conjugate gradient solver are $O(k^3)$ in floorplan [17]. The conjugate gradient solver with preconditioned is $O(k^{2.5})$; in two-dimensional problem, $n = k^2$, and in three-dimensional, $n = k^3$.

The linear equations can be solved by LU in time $2n^3/3$. The analysis and experiment also prove the excellence of the conjugate gradient incremental updating solution method in the SA iterative framework algorithm of thermal floorplan.

5. Results of Experiments

The conjugate gradient algorithm is imported into the open source HotSpot floorplan [15] in C and C++ program language. The HotSpot thermal floorplan uses the SA (Simulated Annealing) [18] optimal algorithm. The experiments are running on Ubuntu with Intel® Core™ CPU i5-2300 2.80 GHz and 12 G memory. The benchmarks are MCNC [19] benchmark circuits. The block power trace is generated by a Perl script random function, and the power density ranges from 10^5 W/m^2 to 10^7 W/m^2 for each block [14].

It is a more convictive way to compare CPU time of two linear solvers in same program; the conjugate gradient algorithm is been implanted by the C++ code and merged into the hot floorplan [15]; and the conjugate gradient algorithm compares with the hot floorplan default LU decomposition solver. The two linear solvers of program are switched by the command line parameter, so the program run environment, such as CPU, memory, GCC version, and compile option, is the same.

Table 1 shows that the conjugate gradient solver without precondition (CG normal in the table) run time is approximate; in the LU decomposition solver on MCNC case, the run time unit is second; the conjugate gradient without precondition solver run time is about speed up averagely 1.49 compared with the LU decomposition solver; the LU decomposition solver once average time is 0.00387 second ($3.87E-03$ in the table); and the conjugate gradient solver without precondition solver once average time is 0.00191 second ($1.91E-03$ in the table).

Table 2 shows that the conjugate gradient solver with Jacobi precondition run time is quicker than the LU decomposition solver, and the conjugate gradient solver with Jacobi precondition run time speeds up averagely 4.32x comparing with the LU decomposition solver. The conjugate gradient solver with the Jacobi precondition solver once had an average time of 0.000567 second ($5.67E-04$ in table).

Table 3 shows that the conjugate gradient solver with SSOR precondition run time is less better than the conjugate gradient solver with the Jacobi Precondition; the conjugate gradient solver with SSOR precondition run time speeds up averagely 5.18x with the LU decomposition solver; the conjugate gradient solver with the SSOR precondition solver once had an average time of 0.000473 second ($4.73E-04$ in table). It is 11x for test case ami49. The profit of conjugate gradient solver with SSOR

```
Require: cost(f) SA floorplan evaluate metric;
Ensure: optimal floorplan f*
 1 r_norm = (r,r);
 2 int Iter = 0;
 3 double α, β, rp, rpold, b_norm, r_norm;
 4 vector r, p, q, z;
 5 initial T schedule;
 6 / * stop annealing if temperature has cooled down enough or max no. of iterations have been tried * /
 7 while (T ≥ Tcold && steps < cfg.Nmax) do
 8    n = cfg.Kmoves * flp− > n units;
 9    i = downs = rejects = 0;
10    sum_cost = 0;
11    /* try enough total or downhill moves per T */
12    while (do(i < 2 * n)&&(downs < n))
13       make random move and data process to floorplan
14       new cost = floorplan evaluate metric;
15       // area (A), temperature (T), and wire length (W):
16       lambdaA * A + lambdaT * T + lambdaW * W
17       reusing the T to be linear solver initial solution
18       if (new cost < cost ‖ rand fraction() < exp(−(new cost − cost)/T)) then
19          /* downhill always accepted */
20          or /* Boltzmann probability function */
21          accepted new cost
22       else
23          rejects++;
24       end if
25       i++;
26    end while
27    /* stop annealing if there are too little accepts */
28    if((rejects/i) > cfg.Rreject)
29          break;
30    /* annealing schedule */
31    T*= cfg.Rcool;;
32    steps++;
33 end while
```

Algorithm 2: The SA thermal floorplan algorithm with the conjugate gradient thermal solver.

Table 1: The conjugate gradient solver without precondition.

Design name	Block number	LU solver			CG		
		Run time	Solver times	Solver average time	Run time	Ratio	Solver average time
xerox	10	5.69	26601	$2.14E-04$	5.32	1.07	$2.00E-04$
hp	11	7.53	29401	$2.56E-04$	8.88	0.85	$3.02E-04$
ami33	33	337.87	91001	$3.71E-03$	299.52	1.13	$3.29E-03$
ami49	49	1534.41	135801	$1.13E-02$	524.65	2.92	$3.86E-03$
avg	25.75	471.38	70701	$3.87E-03$	209.59	1.49	$1.91E-03$

precondition gains the best result ratio comparing with the LU decomposition solver.

Figure 1 shows the conjugate gradient solver run time versus the LU decomposition solver on the MCNC benchmark. The figure name rule: the CG is the conjugate gradient solver without precondition; the Jacobi is the conjugate gradient solver with Jacobi precondition; the SSOR is the conjugate gradient solver with SSOR precondition. The conjugate gradient solver without precondition run time is more than the LU decomposition solver on two small cases, and less than on large cases; the conjugate gradient solver with Jacobi and SSOR precondition experimental results are less than the LU decomposition solver results. The precondition is important for the iterative conjugate gradient solver.

Figure 2 shows the conjugate gradient solver run time ratio versus the LU decomposition solver on the MCNC benchmark. Naming rules are consistent with Figure 1. The experimental ratio curve shows that our iterative conjugate gradient solver accelerated more with increasing number of modules.

We also received the reviewer proposal to adapt to the larger GSRC benchmark of examples to test the scalability of

Table 2: The conjugate gradient solver with the Jacobi precondition.

Design name	Block number	LU solver			CG with Jacobi precondition		
		Run time	Solver times	Solver average time	Run time	Ratio	Solver average time
xerox	10	5.69	26601	$2.14E-04$	3.97	1.43	$1.49E-04$
hp	11	7.53	29401	$2.56E-04$	5.3	1.42	$1.80E-04$
ami33	33	337.87	91001	$3.71E-03$	65.76	5.14	$7.23E-04$
ami49	49	1534.41	135801	$1.13E-02$	165.02	9.3	$1.22E-03$
avg	25.75	471.38	70701	$3.87E-03$	60.01	4.32	$5.67E-04$

Table 3: The conjugate gradient solver with SSOR precondition.

Design name	Block number	LU solver			CG with SSOR precondition		
		Run time	Solver times	Solver average time	Run time	Ratio	Solver average time
xerox	10	5.69	26601	$2.14E-04$	3.74	1.52	$1.41E-04$
hp	11	7.53	29401	$2.56E-04$	5.04	1.5	$1.71E-04$
ami33	33	337.87	91001	$3.71E-03$	53.44	6.32	$5.87E-04$
ami49	49	1534.41	135801	$1.13E-02$	134.7	11.39	$9.92E-04$
avg	25.75	471.38	70701	$3.87E-03$	49.23	5.18	$4.73E-04$

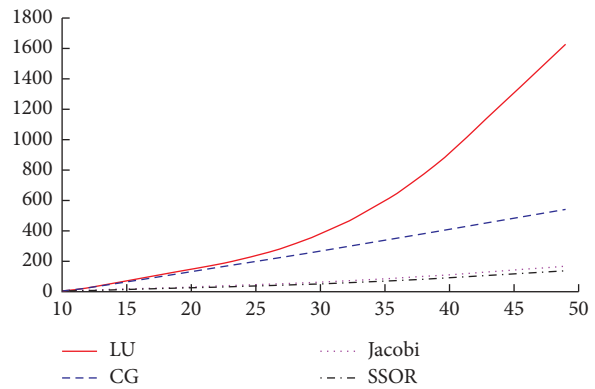


Figure 1: Run time comparing with different block numbers.

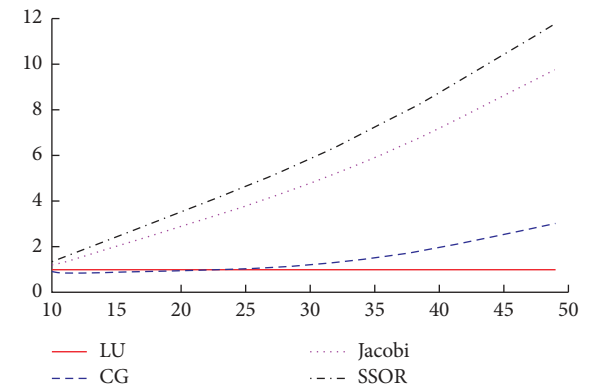


Figure 2: Ratio of run time comparing with different block numbers.

our algorithm. However, we need to emphasize that HotSpot floorplan is designed for the CPU small quantity module floorplan, which is not suitable for running the large collection of placement instances, so the running time is relatively long. Table 4 shows that the conjugate gradient solver with SSOR precondition run time is better than the LU decomposition solver, the conjugate gradient solver with SSOR precondition run time speeds up averagely 17x with the LU decomposition solver, the conjugate gradient solver with SSOR precondition is 24x for test case n300. The profit of the conjugate gradient solver with SSOR precondition gains the best result ratio comparing with the LU decomposition solver.

Table 4: The conjugate gradient solver with SSOR precondition run time on the GSRC benchmark.

Design name	Block number	LU solver second (s)	CG solver with SSOR second (s)	Ratio
n100	100	380.857101	44.486236	8.561234558
n200	200	6411.656280	311.195691	20.60329389
n300	300	32716.53333	1326.933276	24.65574865
Avg	600	13169.68224	560.8717343	17.94009237

6. Conclusions and Future Work

The conjugate gradient solver is often been used in large sparse matrix method computation, and the HotSpot thermal floorplan could be speeded up by using the sparse matrix iterative linear solver.

The experiments show that the iterative conjugate gradient solver is faster than the direct LU decomposition solver on the MCNC benchmark.

The relative sparse matrix theory could be applied to other iterative framework algorithms, and the relative sparse matrix could be extensible. The future works may be the following:

(i) Extend to other precondition methods of an iterative linear solver, and we could use other preconditions.

(ii) Explore sparse linear system theory to speed up our program because there are many theoretical innovations to solve linear system in the last two decades.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

Special thank are due to Michigan Technological University Professor Zhuo Feng for the technical discussion with great significance. This paper was supported by "the Fundamental Research Funds for the Central Universities."

References

[1] Z. H. Jiang, N. Xu, F. Huang, Y. C. Ma, and X. L. Hong, "Hierarchical thermal model using Gauss-Seidel method in floor-planning," in *Proceedings of the International Conference on ASIC (ASICON)*, pp. 2102–2104, Guilin, China, October 2007.

[2] N. Xu and Z. H. Jiang, "Thermal aware floorplanning using Gauss-Seidel method," *Journal of Electronics*, vol. 25, no. 6, pp. 845–851, 2008.

[3] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-trees: a new representation for nonslicing floorplans," in *Proceedings of the Design Automation Conference (DAC)*, pp. 458–463, Los Angeles, CA, USA, 2000.

[4] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, 2003.

[5] Z. H. Jiang, N. Xu, and C. X. Wu, "Thermal floorplan base on conjugate gradient solver in hotspot," in *Proceedings of the International Conference on Resources, Environment, Information and Engineering Innovation (REIEI2014)*, pp. 908–912, Nanchang, China, July 2014.

[6] V. Pavankumar Kalangi, M. Ashwin Kumar, P. Agrawal, R. S. T. G. Srinivasa, and A. Pal, "Methodology for thermal aware physical design," in *Proceedings of the TENCON 2008-2008 IEEE Region 10 Conference*, pp. 1–4, November 2008.

[7] W. Liu, A. Calimera, A. Macii, E. Macii, A. Nannarelli, and M. Poncino, "Layout-driven post-placement techniques for temperature reduction and thermal gradient minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems Year*, vol. 32, no. 3, pp. 406–418, 2013.

[8] M. P.-H. Lin, "Recent research in analog placement considering thermal gradient," in *Proceedings of the 20th European Conference on Circuit Theory and Design (ECCTD)*, pp. 349–352, Linköping, Sweden, August 2011.

[9] J. Song, Y.-M. Lee, and C.-T. Ho, "ThermPL: thermal-aware placement based on thermal contribution and locality," in *Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Hsinchu, Taiwan, April 2016.

[10] M. P.-H. Lin, H. Zhang, M. D. F. Wong, and Y.-W. Chang, "Thermal-driven analog placement considering device matching," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 325–336, 2011.

[11] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M. R. Stan, and W. Huang, "Temperature-aware microarchitecture: modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[12] P. E. Black, "Sparse matrix," in *Dictionary of Algorithms and Data Structures*, vol. 27, U.S. National Institute of Standards and Technology, Gaithersburg, MD, USA, June 2006, http://www.nist.gov/dads/HTML/sparsematrix.html.

[13] K. Michael and T. Heath, *Scientific Computing: An Introductory Survey*, McGraw-Hill, New York, NY, USA, 2nd edition, 2002.

[14] C.-H. Tsai and S.-M. S. Kang, "Cell-level placement for improving substrate thermal distribution," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 19, no. 2, pp. 253–266, 2000.

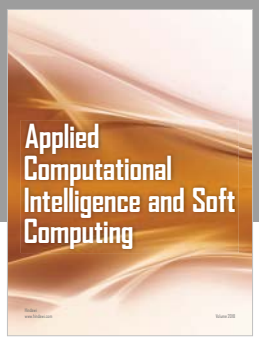
[15] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, *Temperature-Aware Microarchitecture: Extended Discussion and Results*, Tech. Rep. CS-2003-08, Department of Computer Science, University of Virginia, April 2003.

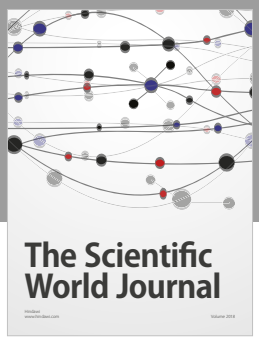
[16] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

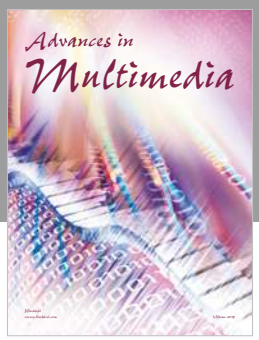
[17] W. Yu, *Numerical Analysis and Algorithms*, Tsinghua University Press, Beijing, China, 2011, in Chinese, http://learn.tsinghua.edu.cn:8080/2003990088/index.htm.

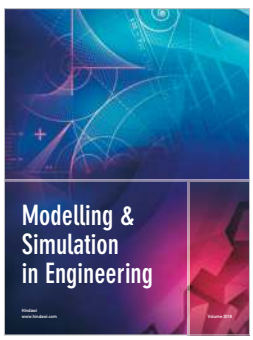
[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

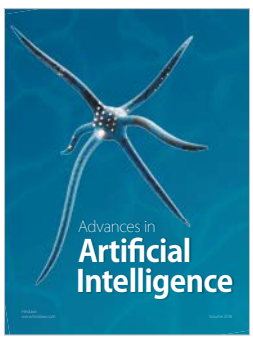
[19] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," in *Proceedings of the ACM/IEEE Design Automation Conference*, Los Angeles, CA, USA, June 1999.

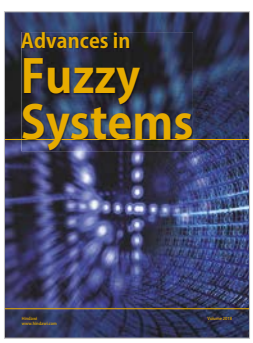












Hindawi
Submit your manuscripts at
www.hindawi.com