

FPGA 并行时序驱动布局算法

张家齐, 沈剑良, 朱 珂

(国家数字交换系统工程技术研究中心, 郑州 450002)

摘 要: 传统的基于模拟退火的现场可编程门阵列(FPGA)时序驱动布局算法在时延代价的计算上存在一定误差,已有的时序优化算法能够改善布局质量,但增加了时耗。针对上述问题,提出一种基于事务内存(TM)的并行FPGA时序布局算法TM_DCP。将退火过程分发至多线程执行,利用TM机制保证共享内存访问的合法性,并将改进的时序优化算法嵌入到事务中并发执行。测试结果表明,与通用布局布线工具相比,8线程下的TM_DCP算法在总线长仅有轻微增加的情况下,关键路径时延平均降低了4.2%,同时获得了1.7倍的加速,且其执行速度随线程数的增加具有较好的可扩展性。

关键词: 现场可编程门阵列;模拟退火算法;并行算法;事务内存;时序驱动布局

中文引用格式: 张家齐, 沈剑良, 朱 珂. FPGA 并行时序驱动布局算法[J]. 计算机工程, 2017, 43(2): 98-104.

英文引用格式: Zhang Jiaqi, Shen Jianliang, Zhu Ke. Parallel Timing-driven Placement Algorithm for FPGA [J]. Computer Engineering, 2017, 43(2): 98-104.

Parallel Timing-driven Placement Algorithm for FPGA

ZHANG Jiaqi, SHEN Jianliang, ZHU Ke

(National Digital Switching System Engineering and Technological Research Center, Zhengzhou 450002, China)

【Abstract】 Traditional timing-driven Field Programmable Gate Array(FPGA) placement algorithm has some degree of error when calculating timing cost. Some timing-driven algorithms achieve better placement quality with a sacrifice of time. To deal with this problem, this paper proposes a timing-driven parallel algorithm TM_DCP based on transactional memory. TM_DCP distributes block swaps into multiple threads, and then uses Transactional Memory(TM) mechanism to ensure the legality of shared memory accesses. An improved timing-driven algorithm is also added in transactions. Experimental results show that compared with Versatile Place and Route(VPR), TM_DCP with 8 threads decreases the Critical Path Delay(CPD) by 4.2% on average with relatively small increase of total wire length. It also achieves 1.7 times speedup, and scales well with the increasing of threads.

【Key words】 Field Programmable Gate Array(FPGA); simulated annealing algorithm; parallel algorithm; Transactional Memory(TM); timing-driven placement

DOI: 10.3969/j.issn.1000-3428.2017.02.017

0 概述

现场可编程门阵列(Field Programmable Gate Array, FPGA)凭借其灵活性和易用性,在当前通信、工业控制等诸多领域得到了广泛应用。为了确保FPGA的工作性能,尽可能减少其效率损失,需要运用专门的时序驱动的布局、布线技术。当前已有的FPGA时序驱动布局算法通常以模拟退火算法为基础,解决时延代价的最优化问题。通用布局布线工具VPR(Versatile Place and Route)^[1]同时以线长和

时延作为代价函数,以得到线网布通率和关键路径时延(Critical Path Delay, CPD)相平衡的布局方案。但传统的基于模拟退火的FPGA时序驱动布局算法在同一温度下不更新关键度信息,使得时延代价的计算存在一定误差,影响代价函数的收敛以及布局结果的优化。文献[2-4]分别针对VPR算法中的时延裕量、关键路径时延、关键度等时序信息进行了设置或修正,牺牲了一定执行效率,得到了时序更优的布局结果。

然而在过去的15年里,FPGA设备尺寸的增速

基金项目: 国家“863”计划重大项目(2014AA01A704);国家自然科学基金(61572520)。

作者简介: 张家齐(1993—),男,硕士,主研方向为数字集成电路设计与分析;沈剑良,助理研究员、博士;朱 珂,副教授、博士。

收稿日期: 2015-11-23 **修回日期:** 2016-01-20 **E-mail:** zhangjqfriend@126.com

几乎是单核 CPU 性能增速的 4 倍^[5], 传统的串行模拟退火算法 (Simulated Annealing Algorithm, SA) 消耗时间往往较长, 加入时序修正后的算法效率已经难以满足布局要求。随着多核处理器的日益普及, 工业界和学术界已经开始将并行思想应用于 SA 布局算法中。文献[6]将每次交换交由多个线程执行, 然而这种方法并行化程度不足, 没有很好的可扩展性。Altera 公司的 Quartus II^[5] 用一个线程负责模块的交换, 同时其他线程并发地对交换进行评估, 使用专门的“独立性检查器”防止交换冲突的产生, 这种方法取得的加速效果有限, 并且代码量较大, 实现较为复杂。TVPR^[7] 并行地进行模块交换, 首次利用了软件事务内存 (Software Transactional Memory, STM) 的方式自动检测并解决冲突的发生, 这种方式实现较为简便、可靠, 具有良好的加速扩展性, 平均只有 1% 的质量降低。相比于其他并行化方法, 事务内存 (Transactional Memory, TM) 机制能极大地简化并行化过程, 但由于 TM 带来的额外开销过大, TVPR 最终的加速效果并不理想。文献[8]基于 STM 实现的 MoveSpec-STM 算法, 细粒化了事务范围, 取得了较好的加速比, 但各方面布局质量都明显下降。

本文在文献[2,4]的基础上使用了新的时序更新策略以及更准确的时序修正方法, 尝试将时序布局算法并行化, 采用 STM 的方式实现共享内存的保护, 将时序优化算法封装入事务中, 提出一种并行的 FPGA 时序布局算法。

1 相关工作

1.1 FPGA 时序驱动布局算法

当前的 FPGA 时序布局算法中, 代表性的是学术界权威的文献[9]采用的时序驱动模拟退火算法。其将代价函数设置为连线长度代价与时序代价的组合函数, 同时顾及总线长和关键路径时延的优化。

在内循环中, 随机地在 R_{lim} 范围内交换可配置逻辑单元的位置, 按照式(1)计算每次交换的代价差:

$$\Delta C = \lambda \frac{\Delta Wiring_Cost}{Pre_Wiring_Cost} + (1 - \lambda) \frac{\Delta Timing_Cost}{Pre_Timing_Cost} \quad (1)$$

其中, $Wiring_Cost$, $Timing_Cost$ 分别代表线长与时延代价; λ 用来调节两者所占权重, 默认为 0.5。算法中判断一次交换是否接受采用 Metropolis 准则, 如式(2)所示。

$$\varepsilon < \exp \frac{-\Delta C}{T} \quad (2)$$

其中, ε 是取值在 $[0, 1)$ 之间的随机数。当 $\Delta C < 0$ 时, 新状态一定会被接受。当 $\Delta C > 0$ 时, 仍会以一

定概率接受新状态: 在初始温度较高的情况下几乎接受所有解, 随着温度降低接受较差解的概率会变得更低, 并且 ΔC 越大接受的概率越低。

在外循环则更新电路的时序信息, 并控制温度及 R_{lim} 的值。其中, 温度初始化为较高温度, 随着退火过程不断降低, 直至“冰点”退出循环; R_{lim} 的初始为整个芯片范围, 随着温度降低会不断变小。

时延代价的计算是通过静态时序分析实现的: VPR 首先将电路转化为一个时序图, 图中的节点代表引脚, 边代表引脚间连接的时延; 然后依据时序图建立起时延裕量 (slack) 矩阵, 进一步得出连接的关键度 $Crit(i, j)$; 最后根据时延 $Delay(i, j)$ 和相应的关键度便可计算出时延代价。关键度和时延代价的计算分别如下:

$$Crit(i, j) = 1 - slack(i, j) / D_{max} \quad (3)$$

$$Timing_Cost(i, j) = Delay(i, j) \times Crit(i, j)^{Crit_Exp} \quad (4)$$

其中, D_{max} 代表当前电路最大时延 CPD; $Crit_Exp$ 是关键度指数。

将每对源端到漏端的时延代价求和, 得到整体布局结果的时序代价, 如式(5)所示。

$$Timing_Cost = \sum Timing_Cost(i, j) \quad (5)$$

由于 VPR 仅当退火温度降低时才重新计算关键度, 关键度却未能得到及时更新, 有可能在判定是否为较优解时出现误差, 导致得到的布局结果不是最优的。通过加入时序优化算法, 可以更精确地反映布局质量的变化, 进一步降低关键路径时延。本文拟使用改进的时序信息优化策略, 以及动态的关键度修正来提高布局的时序质量。

1.2 串程序并行化技术

当前多核处理器日益普及, 然而大多面向单核的串程序无法充分利用多核处理器的优势, 并行编程技术凭借对程序性能的有效提高, 已经受到越来越多的关注。目前广泛应用的并行化技术有: 消息传递接口 (MPI), Pthreads, OpenMP 等。

OpenMP^[10] 是共享内存编程的工业标准 API, 被设计成可以用来对已有的串程序进行增量式的并行化^[11]。OpenMP 提供了“基于指令”的共享内存 API, 利用 #pragma 开头的预处理器指令及一系列控制语句, 只需要对源代码进行少量修改便可时序串程序的并行化。例如: #pragma omp parallel for 指令, 表示将一个 for 循环平均拆分到多个线程去执行; num_threads(n) 语句, 表示设置线程数量为 n 。普通编译器将忽略编译指导语句, 添加编译选项 -fopenmp 时才会生效, 使其有着良好的可移植性。

然而多线程的同步、对共享资源的访问控制是并行程序开发面临的重要问题, 直接关系到算法的

正确性。OpenMP 编译器并不会检查数据依赖及循环依赖关系,仅提供了锁和临界区机制。锁的粒度控制、死锁等问题使得这种方法有很大的局限性,不仅影响并行算法的性能,也使得并程序的开发难度大大增加^[12]。

事务内存的出现为并发访问控制提供了新的思路。TM 能为并行程序提供易用的锁机制,向访问共享内存的任意代码区域提供原子的、连续的操作。TM 机制将程序分割成一个个任务,将每个任务作为一个“事务”分配到多线程执行。由于任务间的独立性无法提前预知,TM 系统会监视所有事务对内存的访问,保证程序的正确性。当多个事务同向同一块内存区域进行写操作时就会产生冲突,此时会将事务“撤销”(ABORT)并“回滚”(ROLL BACK),直到没有冲突产生才会向主内存“提交”(COMMIT)所有的更改,线程间保持相互可见。

TM 根据实现方式的不同又分为硬件事务内存 (HTM) 和软件事务内存 (STM)。其中,STM 将所有功能通过纯代码的方式实现,只需在程序中额外添加一些指令就能自动追踪事务的内存访问,相比于 HTM 有着实现较为简便,不存在容量限制等优势,更便于学术研究;但其有着高额的额外开销,对效率的提升有着一定制约。

本文拟使用 OpenMP + STM 的方式实现时序布局算法的并行化。

2 本文算法

2.1 时序优化算法

在默认情况下,VPR 在内循环中不更新关键度,牺牲了算法精度从而节省时间。因此,可能计算出错误的时延代价差,进而影响布局结果的判断。为平衡该问题,VPR 专门提供了可配置参数 `inner_loop_recompute_divider`,用来调节在内循环中执行时序分析的次数(在 VPR 7.0 中默认为 0),时序分析将对关键度、关键路径时延等进行更新。然而,即时进行时序分析的方法仍可能出现误差。根据式(3)、式(4)可得:

$$\begin{aligned} \text{Timing_Cost}(i,j) &= \text{Delay}(i,j) \times \text{Crit}(i,j)^{\text{Crit_Exponent}} \\ &= \text{Delay}(i,j) \times (1 - \text{slack}(i,j)/D_{\max})^{\text{Crit_Exponent}} \\ &= \text{Delay}(i,j) \times (\text{Path_Delay}(i,j)/D_{\max})^{\text{Crit_Exponent}} \end{aligned} \quad (6)$$

其中, $\text{Path_Delay}(i,j)$ 代表连接 (i,j) 所在路径上的总时延。理想情况下,时延代价 $\text{Timing_Cost}(i,j)$ 的值应随关键路径时延 D_{\max} 单调变化。即当 D_{\max} 提高时, $\text{Timing_Cost}(i,j)$ 应该增加,倾向于拒绝这次交换;当 D_{\max} 减小时, $\text{Timing_Cost}(i,j)$ 应该增加,倾向于接受这次交换。然而,由式(5)可见,当连接 (i,j)

时延发生变化产生新的关键路径时, $\text{Delay}(i,j)$, $\text{Path_Delay}(i,j)$, D_{\max} 三者的值会同时改变,且 $\text{Timing_Cost}(i,j)$ 随 D_{\max} 的变化不单调,这显然会影响布局结果的判断。出现这种情况的主要原因是 D_{\max} 的值发生了改变,即新的关键路径的出现。

为此提出新的时序更新策略:在同一温度下保持 D_{\max} 的值不变,仅当温度降低时更新 D_{\max} ;每次更新时序信息仅修正关键度的值,并允许关键度大于 1。这样,当关键路径上的 $\text{Delay}(i,j)$ 增大时, $\text{Path_Delay}(i,j)$ 随之增大,关键度大于 1, $\text{Timing_Cost}(i,j)$ 值将迅速增大;同理当 $\text{Delay}(i,j)$ 减小时, $\text{Timing_Cost}(i,j)$ 值将迅速减小。这更能鼓励舍弃 CPD 增加的布局,接受 CPD 减小的布局。以 MCNC 测试电路 alu4 为例,测试不同时序分析策略下 CPD 随降温迭代次数的变化,如图 1 所示。图形表示采用原始 VPR 时序分析策略的结果,三角形表示采用新的时序更新策略的结果,都在每个退火温度下进行 100 次时序分析。可以看出,在新的时序更新策略下,CPD 的下降速度更快,并能取得更优的结果,与理论预期保持一致。

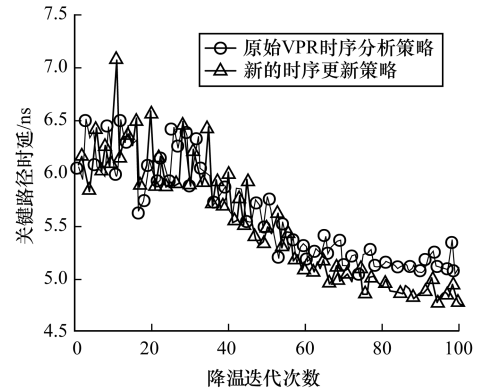


图 1 关键路径时延随降温迭代次数的变化

然而即时的时序分析具有较高的复杂度,故本文采用了动态更新路径关键度的方法,在付出较低的复杂度代价下,尽可能接近完整时序分析的结果:每当有模块交换位置时,仅对发生变化的路径上的关键度信息进行更新;当一个连接的时延大小变化时,包含这条连接的整条路径上的时延都会随之发生改变,关键度也应该随之改变。假设温度 t 时,连接 (k,l) 在第 m 次交换后的时延发生变化,对连接 (k,l) 所在路径上的所有连接 (i,j) ,按照式(7)、式(8)修正其关键度。

$$\begin{aligned} \Delta \text{Crit}(k,l,t,m) &= [\text{Delay}(k,l,t,m) \\ &\quad - \text{Delay}(k,l,t,m-1)]/D_{\max} \end{aligned} \quad (7)$$

$$\text{Crit}(i,j,t,m) = \text{Crit}(i,j,t,m) + \Delta \text{Crit}(k,l,t,m) \quad (8)$$

时延代价函数及增量的计算公式改变为:

$$Timing_Cost(i, j, t, m) = Delay(i, j, t, m) \times Crit(i, j, t, m)^{Crit_Exp} \quad (9)$$

$$\Delta Timing_Cost(i, j, t, m) = [Delay(i, j, t, m) \times Crit(i, j, t, m)^{Crit_Exp}] - [Delay(i, j, t, m-1) \times Crit(i, j, t, m-1)^{Crit_Exp}] \quad (10)$$

时序优化算法的示例如图 2 所示, 其中, A~C 代表电路模块; a~e 代表模块间连接。图 2(a) 表示原始电路, 其时延代价为 13.5, 关键路径为 a-c-e,

CPD 为 10。图 2(b) 为交换模块 C 的位置后按照新的时序更新策略进行时序分析的情况。此时关键路径变为 b-d-e, CPD 变为 12, 比原始电路提高了 2 个单位; 同时计算出时延代价变为 15.9, 比原始电路有所增加, 可以正确判断其为较差的布局。图 2(c) 则表示不进行时序更新, 仅按照式(8)更新其关键度, 时延代价与图 2(b) 重新进行时序分析的结果相比, 仅相差 0.4 个单位, 在计算量大大减少的情况下, 较为精确地反映了布局质量的变化。

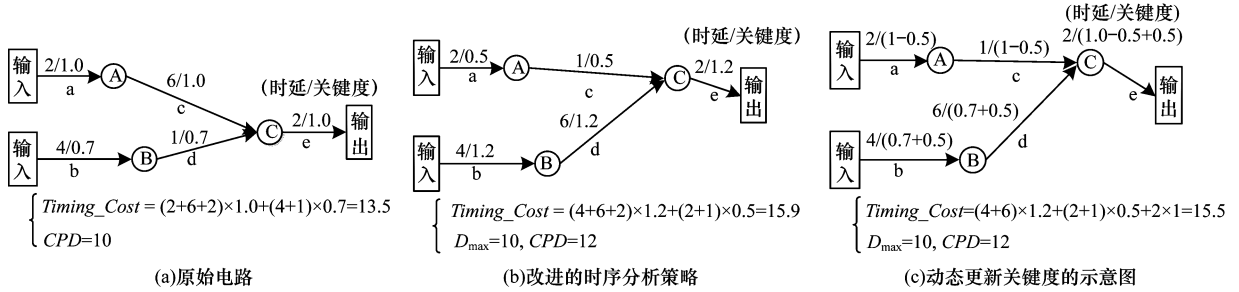


图 2 时序优化示例

与文献[2,4]中的时序修正方法相比,该算法对时延代价的判断有着较高的准确度,更能反映电路的实际变化,从而得到更优的时序结果。

2.2 并行化 VPR 算法

并行冲突问题如图 3 所示。

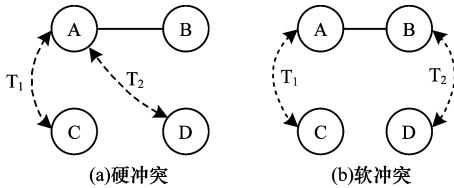


图 3 并行冲突问题

退火算法的内循环进行了大量的模块交换,在布局过程中是耗时较高的步骤,加入时序优化算法也会一定程度上提高复杂度。为了提高算法效率,通过 OpenMP 将模块交换交由多线程并行处理。但由于各个线程间是无法提前预知所要交换模块的布局位置,并行的移动就可能会产生冲突,可分为硬冲突和软冲突:硬冲突是指多个线程的交换涉及到了同一个模块和位置,造成交换失败,如图 3(a) 所示,线程 T_1 和 T_2 同时交换模块 A 便引发了硬冲突;软冲突则是指多个线程的交换涉及到了同一个线网内的模块,由于线网代价差是根据交换前后的布局来计算的,同时改变多个线网内模块的位置将对代价差的计算带来不可预料的误差,可能造成对布局结果的错误判断,如图 3(b) 所示, T_1 希望交换模块 B 的位置, T_2 希望交换模块 A 的位置,而 A, B 同处于

一个线网内,这时便引发了软冲突。

为进一步避免可能存在的冲突,需要对共享内存的读写加以保护。为此加入 STM 机制,目前已有比较成熟的 STM 软件可供使用,其中, TinySTM^[13-14] 是一个以字为同步单位的 C 软件库,只需开发者指定事务边界,并用提供的宏来“注释”共享数据的读写操作即能实现。例如: TM_START(tid, ro), 标记事务的开始, tid 为线程编号, ro 为读写权限; TM_COMMIT, 提交事务, 一般与 TM_START 成对出现; TM_LOAD(addr), 从共享内存 addr 读出数值; TM_STORE(addr, value), 向共享内存 addr 写入数值 value。

首先利用 TM_START 和 TM_COMMIT 宏分别标记事务的开始和结束,将 VPR 中进行交换的函数 try_swap 及添加到其后续的时序优化函数作为一个“事务”进行标记;对有可能产生冲突的数据的读写,采用 TM_LOAD, TM_STORE 等宏注释,每当带注释的读写操作发生时,就会自动调用 STM 库中的 TM 读写操作,不合法的读写将会被撤销。这样便保证了没有 2 组交换会同时访问同一个数据结构。

加入 STM 后,再考虑图 3 的情况:在图 3(a) 中,线程 T_1 和 T_2 希望同时向模块 A 的位置写入新的值, T_1 或 T_2 的事务将被撤销,避免了硬冲突的产生;在图 3(b) 中,假设线程 T_1 先执行完模块 A 的交换后,将重新计算新布局下连接 AB 的线长、时延代价,涉及到对模块 B 的读取,而此时 T_2 正在向模块 B 的位置进行写操作,其中一个事务将被撤销,避免了软冲突的发生。本文算法流程如图 4 所示。

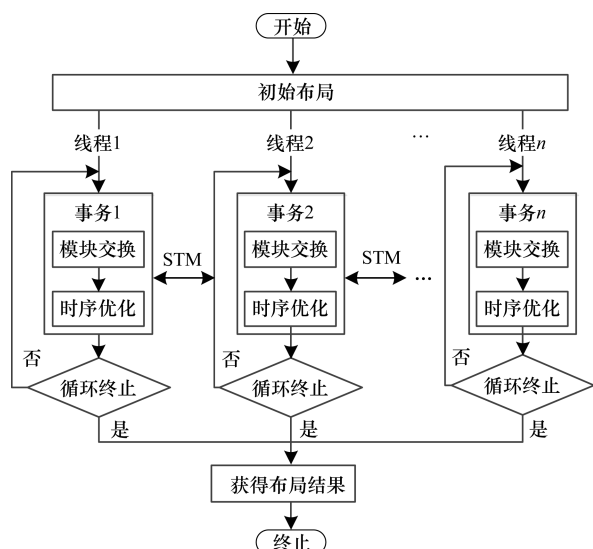


图4 本文算法流程

为进一步提高并行效率,减少冲突发生的可能,对并行算法进行如下优化:

1) VPR 通过生成随机数的方式来选择交换模块,首先使用 OpenMP 提供的 `omp_get_thread_num` 函数获取各线程编号作为各线程的初始种子,利用线性同余随机数生成器为每个线程产生独立的随机数序列,每组随机数队列都相当于一组不同的交换方案,这样线程间对模块交换的选择具有了一定的独立性,减少了冲突的出现。

2) 大规模线网作为较大冲突源(容易引发软冲突),仅将其交由 master 线程(OpenMP 标准中的主线程)处理,避免多线程同时交换同一线网内部模块。本文参考 TVPR^[7] 中的判断方法,将包含模块数大于 10% 总模块数的线网视为大规模线网。

3) 为一些描述电路信息的全局变量(如 block),使用 OpenMP 的 `first private` 语句为每个线程复制一份数据拷贝;对于一些变量的规约操作符的操作(如 `success_sum++`)则利用 `reduction` 规约语句完成,规约语句会自动为变量创建副本并叠加规约操作,有着较高的执行效率。

与简单并行化的 SA 算法相比,优化后的事务撤销率明显下降。使用 `tinySTM` 提供的 `stm_get_stats` 函数获取事务撤销率信息,在电路 alu4 上的测试结果如图 5 所示。

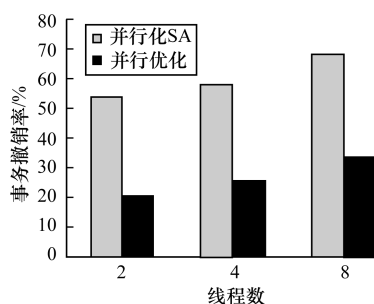


图5 并行算法事务撤销率对比

3 实验结果与分析

实验平台采用开源的 VPR 7.0^[15] 布局布线工具,在其源代码基础上实现了本文算法。仿真电路选用学术界广泛使用的 MCNC 基准电路中最大的 20 组电路,布局架构使用 VPR 7.0 提供的 `k6_frac_N10_mem32K_40nm.xml`,该架构大致对应于现代风格的商用的 40 nm Stratix IV FPGA,包含可分裂的 6-LUT,可配置的 32 KB 内存,以及可分裂的 36×36 乘法器。在 Intel Core i7 3.6 GHz 四核八线程处理器,8 GB 内存,Ubuntu 操作系统的主机上进行仿真。其中,时延代价权重参数 λ 设置为 0.5,关键度指数 `Crit_Exp` 采用默认值,从 1 逐渐增加至 8。VPR 代表最新的 VPR 7.0 算法,TM_DCP 为本文布局算法。

采用 VPR 中集成的 AAPack^[16] 工具将 blif 格式的工艺映射网表进行打包,得到的测试电路的部分信息如表 1 所示。

表1 测试电路信息

电路名	线网数	CLB 模块数	I/O 引脚数
alu4	685	106	22
apex2	1 034	129	41
apex4	707	89	28
bigkey	540	121	426
clma	4 847	534	144
des	1 009	104	501
diffeq	875	93	103
dsip	659	92	426
elliptic	1 885	224	245
ex1010	2 721	316	20
ex5p	676	75	71
frisc	1 874	236	136
misex3	727	96	28
pdc	2 477	336	56
s298	712	122	10
s38417	3 246	392	135
s38584.1	3 380	371	342
seq	954	119	76
spla	1 902	266	62
tseng	609	63	174

在对算法性能进行测试结果如表 2 所示。相比 VPR,布局质量方面,TM_DCP 的总线长平均增加了 2.1%,关键路径时延平均降低了 4.2%,整体结果比较理想;在运行时间方面,TM_DCP 平均降低了 41.3%,即获得了 1.7 倍的加速。

表 2 布局性能测试

电路名	总线长			关键路径时延			运行时间		
	VPR	TM_DCP	比率/%	VPR/ns	TM_DCP/ns	比率/%	VPR/s	TM_DCP/s	比率/%
alu4	120.03	119.39	99.5	5.56	5.22	93.9	8.448	5.040 5	59.7
apex2	179.45	181.40	101.1	6.23	5.99	96.1	11.093	6.275 0	56.6
apex4	123.22	119.48	97.0	5.46	5.14	94.1	8.918	5.009 5	56.2
bigkey	167.44	183.86	109.8	2.39	2.29	95.8	13.504	8.037 0	59.5
clma	1 064.34	1 049.38	98.6	11.59	11.20	96.6	60.880	37.131 5	61.0
des	199.79	214.15	107.2	4.85	4.50	92.8	15.679	10.173 5	64.9
diffeq	103.39	110.19	106.6	6.22	6.08	97.7	8.642	4.921 5	56.9
dsip	158.91	167.35	105.3	2.50	2.52	100.8	11.565	6.079 0	52.6
elliptic	317.92	332.12	104.5	8.69	8.44	97.1	22.964	13.300 5	57.9
ex1010	447.58	458.60	102.5	8.28	7.83	94.6	28.892	16.704 5	57.8
ex5p	108.10	106.50	98.5	5.26	5.07	96.4	7.058	3.835 0	54.3
frisc	373.91	380.18	101.7	11.91	11.72	98.4	26.154	15.342 5	58.7
misex3	124.32	120.91	97.3	4.96	4.62	93.1	8.117	5.027 0	61.9
pdc	619.90	609.47	98.3	8.69	8.24	94.8	30.614	19.741 5	64.5
s298	131.18	136.69	104.2	9.71	9.04	93.1	9.277	6.231 0	67.2
s38417	518.11	549.80	106.1	7.30	6.98	95.6	42.715	24.024 5	56.2
s38584	532.76	558.70	104.9	5.28	5.39	102.1	42.226	23.692 5	56.1
seq	161.68	159.89	98.9	4.99	4.73	94.8	10.790	5.434 0	50.4
spla	421.08	420.25	99.8	7.77	7.17	92.3	24.072	15.413 5	64.0
tseng	57.41	57.78	100.6	6.40	6.09	95.2	5.461	3.108 0	56.9
平均值			102.1			95.8			58.7

接下来测试并行算法的可扩展性,分别设置线程数为 1,2,4,8,测试相对单线程的加速比,如图 6 所示。可见随着线程数的增加,加速效果几乎呈线性增长,证明算法有良好的可扩展性。

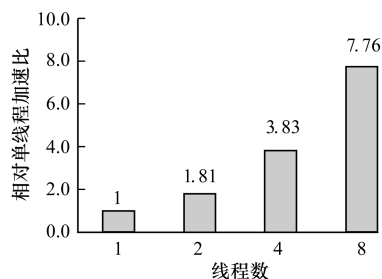


图 6 可扩展性测试

TM_DCP 和其他类似算法的对比如表 3 所示。布局质量方面, TM_DCP 的总线长增加幅度相对较小,关键路径时延均优于其他算法;加速效果方面, TM_DCP 的加速比同样受到了软件事务内存额外开销的制约。相比之下,硬件事务内存 HTM 的方式因

为较小的额外开销而取得了优越的加速比,是未来需要关注的重要技术。

表 3 各并行布局算法相对 VPR 的布局性能对比

算法	线程数	加速比	总线长增量/%	关键路径时延增量/%
TVPR ^[7]	8	0.9	1.6	1.0
MoveSpec-STM ^[8]	16	3.6	8.5	2.3
MoveSpec-HTM ^[8]	4	8.1	19.0	1.7
TM_DCP	8	1.7	2.1	-4.2

4 结束语

本文提出一种并行 FPGA 时序布局算法。首先采用了新的时序分析策略,在每次模块交换后,对其所在路径上所有连接的关键度信息进行修正,得到了更准确的时序信息,然后通过 OpenMP + STM 的编程方法高效地实现了时序布局算法的并行化,并对并行性能进行优化。测试结果表明,该算法能取

得良好的可扩展加速效果,降低关键路径时延。但 STM 的额外开销对加速效果有着较大影响,随着 CPU 对 HTM 支持的普及以及 GPU 编程接口的不断完善,利用 HTM 和 GPU 等方式提升算法效率将是下一步的研究方向。

参考文献

- [1] Betz V, Rose J. VPR: A New Packing, Placement and Routing Tool for FPGA Research [M]//Betz V, Rose J. Field-programmable Logic and Applications. Berlin, Germany: Springer-Verlag, 1997: 213-222.
- [2] Eguro K, Hauck S. Enhancing Timing-driven FPGA Placement for Pipelined Netlists [C]//Proceedings of Design Automation Conference. Washington D. C., USA: IEEE Press, 2008: 34-37.
- [3] 李 鹏, 兰巨龙, 李立春. 统一关键路径时延为基准 FPGA 模拟退火布局算法[J]. 计算机辅助设计与图形学学报, 2011, 23(3): 521-526.
- [4] 任小西, 吴 楚, 丁 宇. 基于 VPR 的 FPGA 布局算法时延改进[J]. 计算机工程, 2014, 40(12): 302-305.
- [5] Ludwin A, Betz V. Efficient and Deterministic Parallel Placement for FPGAs[J]. ACM Transactions on Design Automation of Electronic Systems, 2011, 16(3): 233-250.
- [6] Ludwin A, Betz V, Padalia K. High-quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware [C]//Proceedings of Symposium on Field Programmable Gate Arrays. New York, USA: ACM Press, 2008: 14-23.
- [7] Birk S, Steffan J G, Anderson J H. Parallelizing FPGA Placement Using Transactional Memory [C]//Proceedings of International Conference on Field-programmable Technology. Washington D. C., USA: IEEE Press, 2010: 61-69.
- [8] An M, Steffan J G, Betz V. Speeding Up FPGA Placement: Parallel Algorithms and Methods [C]//Proceedings of the 22nd IEEE Annual International Symposium on Field-programmable Custom Computing Machines. Washington D. C., USA: IEEE Press, 2014: 178-185.
- [9] Marquardt A, Betz V, Rose J. Timing-driven Placement for FPGAs [C]//Proceedings of the 8th International Symposium on Field Programmable Gate Arrays. New York, USA: ACM Press, 2000: 203-213.
- [10] Chapman B, Jost G, van der Pas R. Using OpenMP: Portable Shared Memory Parallel Programming [M]. Cambridge, USA: MIT Press, 2008.
- [11] Pacheco P. An Introduction to Parallel Programming [M]. Amsterdam, Holland: Elsevier, 2011.
- [12] 张 铎. Transactional Memory 优化技术研究 [D]. 长沙: 国防科学技术大学, 2008.
- [13] Felber P, Fetzner C, Marlier P, et al. Time-based Software Transactional Memory [J]. IEEE Transactions on Parallel & Distributed Systems, 2010, 21(12): 1793-1807.
- [14] Felber P, Fetzner C, Riegel T. Dynamic Performance Tuning of Word-based Software Transactional Memory [C]//Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, USA: ACM Press, 2008: 237-246.
- [15] Luu J, Tim L, Betz V. VPR 7.0 Full Release [EB/OL]. (2015-11-05). <http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html>.
- [16] Luu J, Anderson J H, Rose J S. Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect [C]//Proceedings of the 19th International Symposium on Field Programmable Gate Arrays. New York, USA: ACM Press, 2011: 227-236.

编辑 刘 冰

(上接第 97 页)

- [17] Curtis-Maury M, Shah A, Blagojevic F, et al. Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores [C]//Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York, USA: ACM Press, 2008: 250-259.
- [18] CUDA Toolkit Documentation [EB/OL]. [2015-09-01]. <http://docs.nvidia.com/cuda/cupti/index.html#axzz40mOWtBD7>.
- [19] Eberly L E. Multiple Linear Regression [J]. Topics in Biostatistics, 2007, 23(2): 165-187.
- [20] 徐勇军, 陈静华, 骆祖莹, 等. CMOS 电路动静态功耗协同分析[J]. 计算机工程, 2006, 32(10): 231-233.
- [21] 田景文, 高美娟. 人工神经网络算法研究及应用 [M]. 北京: 北京理工大学出版社, 2006.
- [22] Golub G H, Heath M, Wahba G. Generalized Cross-validation as a Method for Choosing a Good Ridge Parameter [J]. Technometrics, 1979, 21(2): 215-223.

编辑 陆燕菲