软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(6):2724−2752 [doi: 10.13328/j.cnki.jos.007101]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向全分布式智能建筑系统应用程序的并行化编译方法*

陈文杰[1], 杨启亮[1], 姜子炎[2], 邢建春[1], 周启臻[1], 邹荣伟[1], 冯博伟[1]

[1](陆军工程大学 国防工程学院, 江苏 南京 211101)

[2](清华大学 建筑节能研究中心, 北京 100084)

通信作者: 杨启亮, E-mail: yql@893.com.cn

摘　要: 群体智能系统通过邻居个体的信息交互实现群体级别的应用任务, 具有良好的鲁棒性和灵活性. 与此同时, 大多数开发人员难以对分布式、并行的个体交互机制进行描述. 一些高级语言允许用户以串行思维方式、从系统全局角度来编程并行的群体智能计算任务, 而无需考虑通信协议、数据分布等底层交互细节. 但面向用户、全局声明式的群体智能系统应用程序与个体并行执行逻辑存在的巨大语义差距, 使得编译过程复杂进而导致应用程序开发效率不高. 提出一个编译系统及其支撑工具, 支持将高级的群体智能系统应用程序转换为安全、高效的分布式实现. 所提编译系统通过并行信息识别, 计算划分, 交互信息生成技术, 将面向系统全局、串行编程的群体智能应用程序编译为面向个体独立执行的并行目标代码, 从而使用户不必了解个体间的复杂交互机制. 设计一种标准化中间表示, 将复杂群体智能计算任务转换为群体智能算子和输入输出变量组合而成的标准化语义模块序列, 其以独立于平台的形式表示源程序信息, 屏蔽目标硬件平台的异构性. 在一个群体智能系统案例平台中部署和测试所提编译系统, 结果表明该系统能够有效将群体智能应用程序编译为平台可执行的目标代码并提升应用程序开发效率, 其生成的代码在一系列基准测试中具有比现有编译器更好的性能.

关键词: 群体智能; 应用程序; 编译系统; 并行化编译

中图法分类号: TP314

中文引用格式: 陈文杰, 杨启亮, 姜子炎, 邢建春, 周启臻, 邹荣伟, 冯博伟. 面向全分布式智能建筑系统应用程序的并行化编译方法. 软件学报, 2024, 35(6): 2724–2752. http://www.jos.org.cn/1000-9825/7101.htm

英文引用格式: Chen WJ, Yang QL, Jiang ZY, Xing JC, Zhou QZ, Zou RW, Feng BW. Parallel Compilation Method for Fully Distributed Intelligent Building System Applications. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2724–2752 (in Chinese). http://www.jos.org.cn/1000-9825/7101.htm

Parallel Compilation Method for Fully Distributed Intelligent Building System Applications

CHEN Wen-Jie[1], YANG Qi-Liang[1], JIANG Zi-Yan[2], XING Jian-Chun[1], ZHOU Qi-Zhen[1], ZOU Rong-Wei[1], FENG Bo-Wei[1]

[1](College of Defense Engineering, Army Engineering University of PLA, Nanjing 211101, China)

[2](Building Energy Conservation Research Center, Tsinghua University, Beijing 100084, China)

Abstract: Swarm intelligence systems realize group-level application tasks by information interaction of individual neighbors, and have sound robustness and flexibility. Meanwhile, most developers struggle to describe distributed and parallel individual interaction mechanisms. Some high-level languages allow users to program parallel swarm intelligence computing tasks in a serial mindset and from a global system perspective, without considering low-level interaction details such as communication protocols and data distribution. However, the huge semantic gap between user-oriented, globally declarative swarm intelligence system applications and individual parallel execution logic makes the compilation process complex and application development inefficient. Thus, this study proposes a compilation

* 基金项目: 国家自然科学基金 (52178307); 江苏省自然科学基金 (BK20201335); 国家重点研发计划 (2017YFC0704100)
本文由“编译技术与编译器设计”专题特约编辑冯晓兵研究员、郝丹教授、高耀清博士、左志强副教授推荐.
收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-03-29

© 中国科学院软件研究所 http://www.jos.org.cn

system and its supporting tools to support the conversion of high-level swarm intelligence system applications into secure and efficient distributed implementations. By parallel information identification, computing division, and interactive information generation, the compilation system compiles the swarm intelligence application program for global and serial programming into parallel object code for individual execution, and thus users do not have to understand the complex interaction mechanism among individuals. Additionally, a standardized intermediate representation of the compilation system is designed to convert complex swarm intelligence computing tasks into a standardized semantic module sequence composed of swarm intelligence operators and input and output variables, which represents source program information in a platform-independent form and shields the heterogeneity of target hardware platforms. The system is deployed and tested in a case platform of swarm intelligence systems. The results show that the compilation system can compile swarm intelligence applications into platform-executable object code and improve the application development efficiency, and its generated code has better performance than existing compilers in a series of benchmarks.

Key words: swarm intelligence; application; compilation system; parallel compilation

群体智能[1]是指由一定规模的个体通过相互协作在整个群体系统宏观层面表现出来的一种分散、去中心化的自组织行为, 已被广泛应用至智能建筑[2], 物联网[3], 机器人群任务规划[4]等领域. 与集中式方法相比, 群体智能系统具有更好的故障健壮性和并行性[5], 但代价是更复杂的设计. 开发者需要从每个个体执行的角度进行并行编程, 并考虑个体间通信协议、数据分布等底层交互细节, 这导致较高的编程专业门槛和编程成本.

一些研究提出用于开发群体智能系统的编程语言以降低编程难度. SwarmL[6]是一种面向建筑用户的编程语言, 其支持用户以串行思维方式从全局角度编程并行化的全分布式智能建筑系统应用程序. Buzz 编程语言[7]允许开发人员将复杂的群体机器人算法构建为定义良好、可重用的组件. Solidity[8]是一种为实现区块链智能合约而创建的高级编程语言, 其专注于围绕数据和对象而不是逻辑和功能进行编程, 对用户更加友好. 这些语言支持将群体智能系统作为一个统一的整体进行编程, 使得用户从处理个体间交互和通信等底层细节中解脱出来, 降低了群体智能系统的开发和部署难度.

然而, 高级的群体智能系统应用程序到可执行目标代码的编译过程面临巨大挑战, 进而直接影响应用程序开发效率. 首先, 用于编程群体智能系统的高级语言大多贴近用户串行思维方式, 从系统全局的角度来编程分布式、并行的群体智能应用任务, 而在应用程序运行阶段, 个体仅从自身角度独立执行代码, 考虑自身与哪些邻居交互数据以及如何交互. 因此, 串行、面向全局的群体智能系统应用程序与面向个体单独执行的目标代码存在很大的语义差距. 第二, 用户通过对特定数据结构的读写或者函数组合隐式地规范了同步和通信, 因此相邻个体间的变量读写机制、并行任务管理、分布式计算属性等复杂底层细节完全留给编译器和运行时系统. 由于应用程序可能需要在大规模群体智能系统上执行, 这种复杂的协调逻辑必须高效和可伸缩的实现. 第三, 由于群体智能系统硬件平台的异构性, 用户需要就不同架构的硬件平台分别编写程序代码并分别编译, 增加了软件开发难度. 群体智能系统应用程序和底层硬件指令的升级和更新也将给编译系统的维护带来巨大的工作量.

为降低群体智能系统和多核处理器系统软件编译的复杂度, 提高并行计算机系统硬件资源利用率, 国内外学者针对自动并行化编译系统做了大量研究工作[9,10]. 首先, 在编译面向全局和声明式编程的群体智能应用程序方面, 面向多机器人系统应用程序[11]和 Meld 语言[12]的编译器支持将遵循逻辑编程范式的集成级机器人应用程序编译成可以在多个计算单元中单独执行的分布式代码. 另一方面, 很多流行的自动并行化框架[13,14]被相继提出, 例如 Intel Compiler[15]和 PGI[16]等知名商业编译器, 以及 Cetus[17], ROSE[18], TRACO[19], Pluto[20]等学术方法. 这些工具接收源代码作为输入, 并根据目标编程模型 (例如 OpenMP) 生成适合并行体系结构运行的并行程序. 然而, 由于难以充分利用程序的静态信息和缺乏建模精度, 自动并行化生成的代码可能会导致并行化开销, 进而表现出较差的性能. 第二, 在生成支持群体智能系统个体间通信的代码方面, Paraguin[21]提出典型的面向分布式存储系统 MPI 的自动并行化编译系统, 其以计算划分为中心, 通过依赖关系测试确定并行化循环集, 并生成消息传递代码. 然而, 基于 MPI 的分布式系统中的个体采用广播/收集的方式与所有个体进行通信[22], 群体智能系统仅支持个体与相邻个体交互通信来实现全局计算任务, 而不依靠全局地址[2]. 个体交互和通信模式的差异使得面向 MPI 的消息传递代码生成算法难以适用于群体智能应用程序编译系统. 第三, 相关研究提出通用化、平台无关的中间表示 (IR) 来屏蔽目标平台的异构性. INSPIRE[23]使用一组统一的、封闭的中间语言结构对 OpenMP、Cilk、OpenCL 和 MPI 原

© 中国科学院软件研究所 http://www.jos.org.cn

语进行建模, 为并行代码建立可扩展的编译器基础结构. Tapir[24]将程序的控制流表示为与语言无关的控制流图(CFG), 并在程序的 CFG 中不对称地表示逻辑 fork-join 并行性. STC 编译器[25]提供了一种扁平的、易于分析的 IR 来捕获数据驱动的任务并行性执行模型. 每个 IR 过程都被构造成一个块树, 每个块表示为语句序列. 尽管上述 IR 能够以独立于平台的形式表征源程序, 然而, 由于其并没有考虑邻居交互等群体智能系统计算特征, 难以直接应用至群体智能应用程序的编译过程.

针对现有研究存在的问题, 本文以一种用于编程全分布式智能建筑系统的 SwarmL 语言为例, 提出面向群体智能系统应用程序的编译系统, 并实现其支撑工具. SwarmL 是一种高级语言, 支持用户以串行思维方式高效编程并行化的建筑控制应用程序. 本文具体贡献如下.

(1) 提出一种面向群体智能系统应用程序的编译系统及其支撑工具. 通过并行信息识别、任务划分、数据分配等技术, 该系统将符合用户串行思维方式、面向全局角度编程的群体智能系统应用程序解耦、分离为面向单个节点独立执行的中间代码. 根据源程序的隐式并行语义编译生成变量读写机制、对邻居节点的循环遍历操作以及分布式计算属性, 以支持群体智能系统底层节点间的通信和协作. 实现了具有程序编辑和代码生成功能的可视化编译系统工具, 将复杂工作量由用户编程转移到编译和运行时系统, 显著降低了应用程序开发难度.

(2) 提出一种通用化、标准化的编译系统中间表示. 在分析群体智能系统体系结构和程序执行模型的基础上, 该中间表示将每个群体智能操作表示为输入输出变量和标准化群体智能算子相结合的计算事件, 因而具有并行和分布式特征的群体智能应用程序被转换为按照语义顺序排列而成的计算事件序列. 该中间表示能够适配一类具有节点交互协作、并行执行特征的群体智能系统硬件平台, 进而屏蔽硬件平台的异构性.

(3) 提供更好的应用程序开发效率和目标代码执行性能. 以全分布式智能建筑系统为例对该编译系统进行测试, 结果表明, 该编译系统能够将串行的群体智能系统应用程序转换为硬件平台可执行的并行目标代码, 且提升应用程序开发效率, 其生成的代码在一系列基准测试中具有比现有编译器更好的性能.

第 1 节介绍群体智能系统应用程序的一个案例并总结编译应用程序的重要挑战. 第 2 节提出面向群体智能系统应用程序的编译系统总体框架和中间表示的标准化格式. 第 3 节形式化建模群体智能系统应用程序的具体编译算法并研究实现编译系统工具. 第 4 节通过实验验证该编译系统的有效性和效率. 第 5 节讨论群体智能系统程序编译和程序自动并行化方法的相关工作, 并与本文方法相比较. 第 6 节进行总结与展望.

1 群体智能系统应用程序编译挑战: 以 SwarmL 语言为例

本节介绍全分布式智能建筑系统及其编程语言 SwarmL, 其是群体智能系统的一个典型案例. 以 SwarmL 所开发的分布式建筑控制应用程序为例, 总结编译群体智能系统应用程序的关键挑战.

1.1 全分布式智能建筑系统

全分布式智能建筑系统仿照自然界中蜜蜂、蚂蚁等群体的工作机制, 将建筑建模为由一组智能节点组成的无中心、分散的计算网络[2,26], 如图 1 所示. 每个节点具有计算和通信功能, 负责控制和管理本地的一个建筑空间或一件大型机电设备. 节点配置有分布式操作系统[27]和一个标准信息模型[28], 存储了对应的建筑空间或机电设备的各类参数信息. 节点仅与直接相邻的节点进行数据交互, 从邻居节点处获取输入信息, 完成本地计算, 再将计算结果传送给邻居节点. 所有节点以一种自组织的协作机制相互配合完成复杂的建筑控制和管理任务, 这避免了集中式系统存在的中央链路拥堵、配置复杂等问题, 具有即插即用、自组织、自适应等特征[29].

基于标准化的建筑信息模型和智能节点, 建筑功能系统的定义变得标准化和通用化, 其构建和部署过程也变得简单. 全分布式智能建筑系统将建筑物理场模型和分布式、并行计算模式深度融合, 允许用户将建筑管理控制任务开发为通用化应用程序 (application, APP) 并下载到智能节点网络上执行, 例如变风量空调系统优化控制[30]、冷机群控[31]等. 这些应用程序比传统的集中式建筑系统控制策略表现出更好的鲁棒性和收敛性.

1.2 SwarmL

全分布式智能建筑作为群体智能系统的一个实例, 其编程语言 SwarmL 能够直观刻画典型群体智能控制计算逻辑, 因此, 本文选择 SwarmL 语言作为案例来描述群体智能系统应用程序的编译过程.

© 中国科学院软件研究所 http://www.jos.org.cn

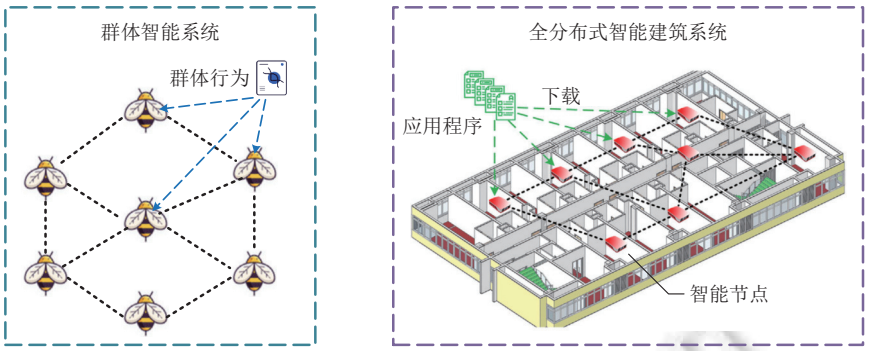


图 1 全分布式智能建筑系统架构

SwarmL 是用于开发全分布式智能建筑系统应用程序的编程语言, 简单而富有表现力. 其建立了用于描述建筑控制任务中的建筑物理对象及其动态执行机制的高级编程抽象, 支持用户以串行思维方式和全局视角编程并行化的全分布式智能建筑系统应用程序, 而无需考虑通信协议和数据分布等底层细节. 主要包含以下编程概念.

● 节点 (node) 对应于群体智能系统中的个体, 是构成全分布式智能建筑系统的基本单元. 应用程序的部署执行以及数据的存储和处理都在各个节点上分布完成. 根据所管理的建筑单元类型和功能差异, 全分布式智能建筑系统中包含不同类型的节点, 如房间, 空调箱, 冷机等.

● 域 (domain) 对应于全分布式智能建筑系统中的各类建筑功能系统, 例如变风量空调系统, 水泵系统等. 域表示若干节点的集合及连接这些节点的拓扑关系, 其可以指定某段应用程序的执行范围, 通过域中节点的相互协作完成特定建筑控制任务.

● 场变量 (field-oriented variable) 是某个有具体含义的建筑物理量在全分布式智能建筑系统中的分布场, 即该量在若干节点上的分布数据集合. 场变量对应于全分布式智能建筑系统中的各类建筑物理信息, 例如温度, 湿度等. 在场变量定义语句中, 节点类型和域名指定了场变量的有效数值的分布范围.

● 参数 (parameter) 是场变量的数据属性, 表示场变量在计算域节点集合上的数据投影, 其能够直接代入表达式或函数进行运算. 通过对参数的定义, 可以根据计算需求灵活选择将哪些节点范围内的场变量数值代入计算表达式. SwarmL 语言支持用户从网络全局和邻域角度分别编程全分布式智能建筑系统控制任务, 既能声明式、粗粒度的描述网络全局计算任务, 也能细粒度的描述本节点与邻居的交互行为.

● 动态计算域 (computational scope) 是在描述群体智能计算操作时用到的一些具有共同功能角色属性的节点集合, 并作为界定场变量映射到参数的对应关系的通用描述方法, 其显式定义了应用程序执行所涉及的节点范围. 用户可以根据计算域向符合特定条件的节点分配计算任务, 进而直观表达复杂的群体智能算法.

● Swarm 并行语句 (Swarm parallel statement) 通过细粒度描述本节点和邻居节点的信息交互实现网络全局层面的并行计算逻辑. 由某个节点作为发起点触发计算任务后, 网络中各个节点都执行相同的计算逻辑, 并行处理 Swarm 并行语句中的各条子语句. Swarm 并行语句中出现的参数都是局部参数.

后文图 2 给出 SwarmL 关键语言元素的类图, 具体语法请参考附录中的图 A1.

1.3 基于 SwarmL 的典型群体智能控制算法描述

本节介绍 3 个利用 SwarmL 开发的全分布式智能建筑系统应用程序段, 来展示 SwarmL 如何简洁编程分布式并行的群体智能系统应用程序, 进而分析编译群体智能系统应用程序所面临的挑战.

(1) 全局扩散任务

全分布式智能建筑系统中的人员疏散系统控制任务将火灾报警信号从发生火灾的区域扩散到所有空间区域[29]. 在该操作中, 由发生火灾的节点作为发起点触发计算任务, 并将报警信号发送给它的邻居, 经过层层传递, 所有节点都收到火灾报警信号. 通常, 编程用户需要从单个节点的角度描述节点如何与邻居交互数据, 并考虑消息发送、接收等底层通信细节. 而 SwarmL 允许用户从系统全局角度来声明式编程并行的扩散任务, 这符合用户的

© 中国科学院软件研究所 http://www.jos.org.cn

串行思维. 在图 3(a) 中, 计算域为 StartNode (发起点) 的全局参数 FireSource, 和计算域为 All (所有节点) 的全局参数 Alarm, 是火灾报警信号对应的场变量在发起点和所有节点上的数据投影. 图 3(a) 第 5 行的全局参数表达式直接将 FireSource 的值赋值给 Alarm, 实现了火灾报警信号从发起点扩散到人员疏散系统中的所有节点.

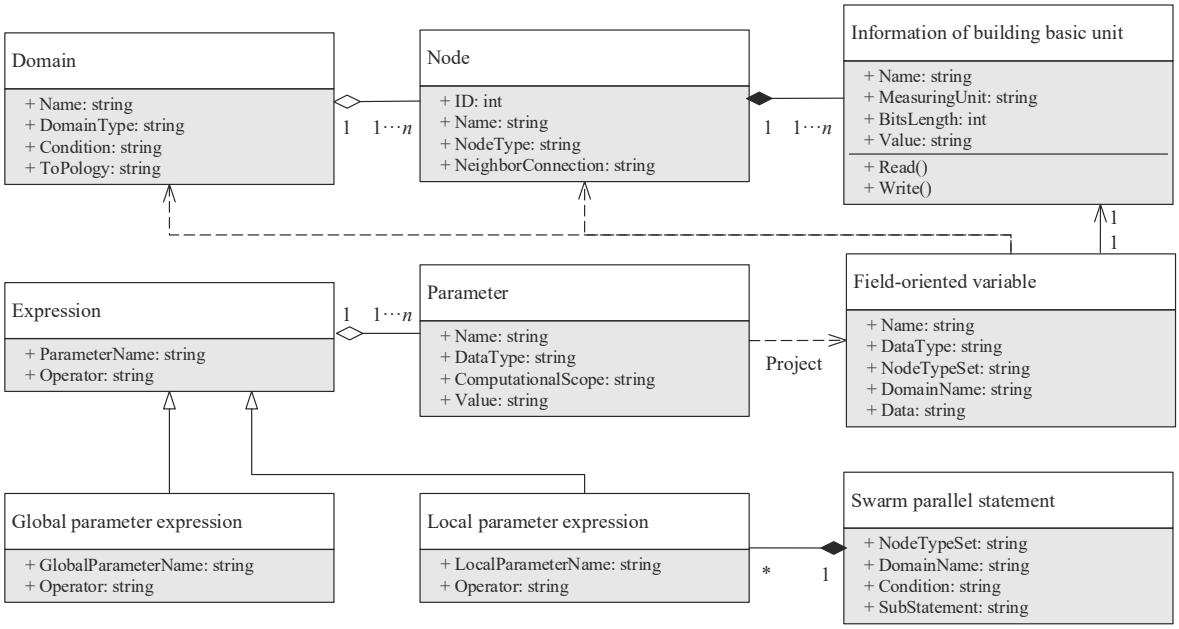
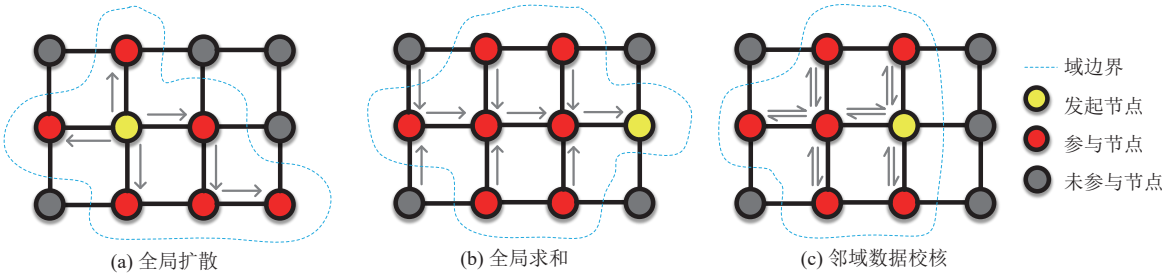


图 2 SwarmL 关键语言元素类图



(a) 全局扩散 (b) 全局求和 (c) 邻域数据校核

(a)

```
1 VPN EvacuationSystem {NodeType: ROOM;}                                  // 定义人员疏散功能系统对应的域
2 VAR FireAlarm {NodeType: ROOM; VPN: EvacuationSystem; DataType: bool;}  // 定义火灾报警信号对应的场变量
3 bool FireSource (Refer to: FireAlarm @global: StartNode);               // 定义表示火灾报警信号的全局参数, 计算域为发起点
4 bool Alarm (Refer to: FireAlarm @global: All);                          // 定义表示火灾报警信号的全局参数, 计算域为所有节点
5 Alarm = FireSource;                                                     // 将火灾报警信号从发起点扩散到域中所有节点
```

(b)

```
1 VPN VAVsystem {NodeType: [ROOM, AHU];}                                  // 定义变风量空调系统对应的域
2 VAR AirVolume {NodeType: ROOM; VPN: VAVsystem; DataType: float;}        // 定义房间需求风量对应的场变量
3 float AirVolume_Sum (Refer to: AirVolume @global: StartNode);           // 定义表示房间需求风量之和的全局参数
4 float AirVolume_All (Refer to: AirVolume @global: All);                 // 定义表示每个房间需求风量的全局参数
5 AirVolume_Sum = +AirVolume_All;                                         // 发起点求取所有房间需求风量之和
```

(c)

```
1 VAR damperOpen_Correction {NodeType: ROOM; VPN: VAVsystem; Datatype: float;}  // 定义房间风阀开度变化量的校核值对应的场变量
2 Swarm (Nodetype: [ROOM]; VPN: [VAVsystem])                                    // 定义Swarm语句实现邻域并行计算
3 {     float dOpen_Cor_0(Refer to: damperOpen_Correction @local: Me);          // 定义表示风阀开度校核值的局部参数, 计算域为本节点
4       float dOpen_Cor_U(Refer to: damperOpen_Correction @local: Upstream);    // 定义表示风阀开度校核值的局部参数, 计算域为上游节点
5       dOpen_Cor_0 = max(dOpen_Cor_U)*delta_0;  }                              // 邻域风阀风阀开度校核运算
```

图 3 基于 SwarmL 的典型群体智能控制逻辑描述

(2) 全局求和任务

在全分布式智能建筑系统中的变风量系统优化控制任务中, 空气处理机组 (AHU) 节点计算变风量系统域中所有房间节点的需求风量之和[26]. 由 AHU 节点作为发起点触发计算任务, 并作为根节点建立生成树, 每个节点将下游邻居节点的需求风量的值与自身的需求风量值求和并传给上游邻居节点, 最终 AHU 节点求得系统中需求风量之和. SwarmL 允许用户使用以包含单个运算符的全局参数表达式简洁描述该全局求和任务. 在图 3(b) 中, 计算域为 StartNode 的全局参数 AirVolume_Sum, 和计算域为 All 的全局参数 AirVolume_All, 是房间需求风量对应的场变量在发起点和所有节点上的数据投影. 图 3(b) 第 5 行的全局参数表达式能够直接求取所有房间节点的需求风量之和并赋值给发起点上的 AirVolume_Sum 参数.

SwarmL 是一种数据并行语言, 其定义的场变量和参数本质上都具有并行语义. 可以看出 SwarmL 能够将分布式建筑控制任务的系统级功能需求直观映射到单个节点的行为, 不需要用户深入理解和描述节点之间的通讯协议、数据分布等底层细节, 降低了编程难度.

(3) 邻域数据校核任务

在变风量系统优化控制任务中, 当某个房间的风阀开度变化量的预期值发生变化, 将触发风阀开度校核任务, 该房间作为根节点构建生成树. 每个房间节点将生成树中上游邻居传来的风阀开度变化量的校核值的最大值乘以耦合系数, 得到本地风阀开度变化量的校核值, 并传递给下游邻居, 每个节点都并行执行该计算逻辑从而实现系统中所有节点的风阀开度校核. 由于该任务细粒度的描述了全分布式智能建筑系统中节点与邻居的数据交互行为, 这难以从全局角度进行编程. SwarmL 提供的 Swarm 并行语句支持用户从单个节点及其邻居的角度描述并行的邻居风阀校核任务. 在图 3(c) 中, 计算域为 Me (本节点) 的局部参数 dOpen_Cor_0, 和计算域为 UpStream (下游节点) 的局部参数 dOpen_Cor_U, 分别表示房间风阀开度变化量的校核值对应的场变量在本地和上游邻居的数据投影. 图 3(c) 第 5 行的局部参数表达式将收取到的上游邻居的校核值取最大, 并乘以相邻风阀开度变化耦合系数, 赋值给本地的风阀开度变化量的校核值. 变风量系统中所有房间节点都并行执行此程序, 直至所有节点都计算得到本地的校核值.

Swarm 并行语句对所有邻居的循环、遍历操作是隐式的, 直观刻画节点和邻居的数据交互过程. 计算域提供了通用化的网络拓扑关系描述方法. 通过声明式指定计算参数和计算所涉及的节点范围, SwarmL 程序可以部署至不同结构的建筑功能系统中, 而不是只面向某一特定的建筑结构.

1.4 编译群体智能系统应用程序的关键挑战

以 SwarmL 开发的全分布式智能建筑系统应用程序为例, 本节总结了编译群体智能系统应用程序的关键挑战.

(1) 如何将以用户串行思维开发的群体智能应用程序解耦、分解为节点可独立执行的目标代码?

SwarmL 是一种声明式语言. 在应用程序开发阶段, SwarmL 支持用户将系统整体作为编程对象, 以串行思维方式编程分布式、并行的群体智能计算任务, 降低了编程难度. 而在运行阶段, 应用程序需要以分布式方式在多个智能节点上执行. 智能节点仅从自身角度执行节点级的程序, 考虑自身与哪些邻居节点交互数据以及如何交互. 因此, 面向用户串行思维和系统全局编程的高级应用程序与底层个体可执行的节点级代码之间存在显著语义差异, 这导致编译过程复杂.

(2) 如何将从邻域角度声明式编程的 Swarm 并行语句转换为支持底层节点并行交互的目标代码?

Swarm 并行语句通过声明式指定参与运算的邻居节点范围和局部参数来描述邻域交互逻辑. 其利用局部参数编程多个邻居节点参与的运算, 对所有邻居的循环、遍历操作是隐式的, 且屏蔽了节点通信协议、数据分布等底层细节. 因此, 节点间的数据移动和并行任务管理等细节完全留给语言的编译和运行时系统. 然而在应用程序运行阶段, 节点需要显式确定每个参数的读写属性, 建立节点与邻居的通信关系以实现对邻居数据的循环遍历操作, 还需要确定计算任务如何触发、计算结果如何发布等分布式计算属性. 因此, 如何将从邻域角度编程的 Swarm 并行语句转换为目标代码, 恢复分布式计算底层细节, 是重要的编译挑战.

© 中国科学院软件研究所 http://www.jos.org.cn

(3) 如何设计编译系统, 使其适应目标硬件平台的异构性和升级更新?

随着群体智能思想的日益普及, 具有节点交互和协作特征的底层硬件控制器可能来自不同厂家, 具有不同的硬件结构和内存资源. 由于底层硬件平台的异构性, 用户需要就不同架构的硬件平台分别编写程序代码并分别编译, 这导致复杂的软件开发过程和跨平台移植困难. 另外, 群体智能系统应用程序和底层硬件指令的升级和更新将给编译系统的维护带来巨大的工作量. 因此, 编译系统应该与特定硬件平台解耦, 且适应应用程序和底层硬件的升级更新, 为上层用户开发提供方便.

2 编译系统定义

针对上述编译挑战, 本文提出一种面向群体智能应用程序的编译系统, 旨在将面向全局、串行思维编程的群体智能系统应用程序编译转换为节点可独立执行的目标代码, 且屏蔽目标硬件平台的异构性. 本节建立了编译系统的总体框架, 并形式化定义编译系统的中间表示接口.

2.1 编译系统总体框架

所提出的编译系统包括解析层, 中间层和映射层, 如图 4 所示. 其中, 解析层面向群体智能系统应用程序, 映射层面向异构的群体智能系统硬件平台. 中间层由标准化语义模块和逻辑链组成, 为上层应用程序和底层硬件平台提供统一的标准接口. 对于不同的硬件平台和操作系统, 中间层可以有多个符合接口的实现.

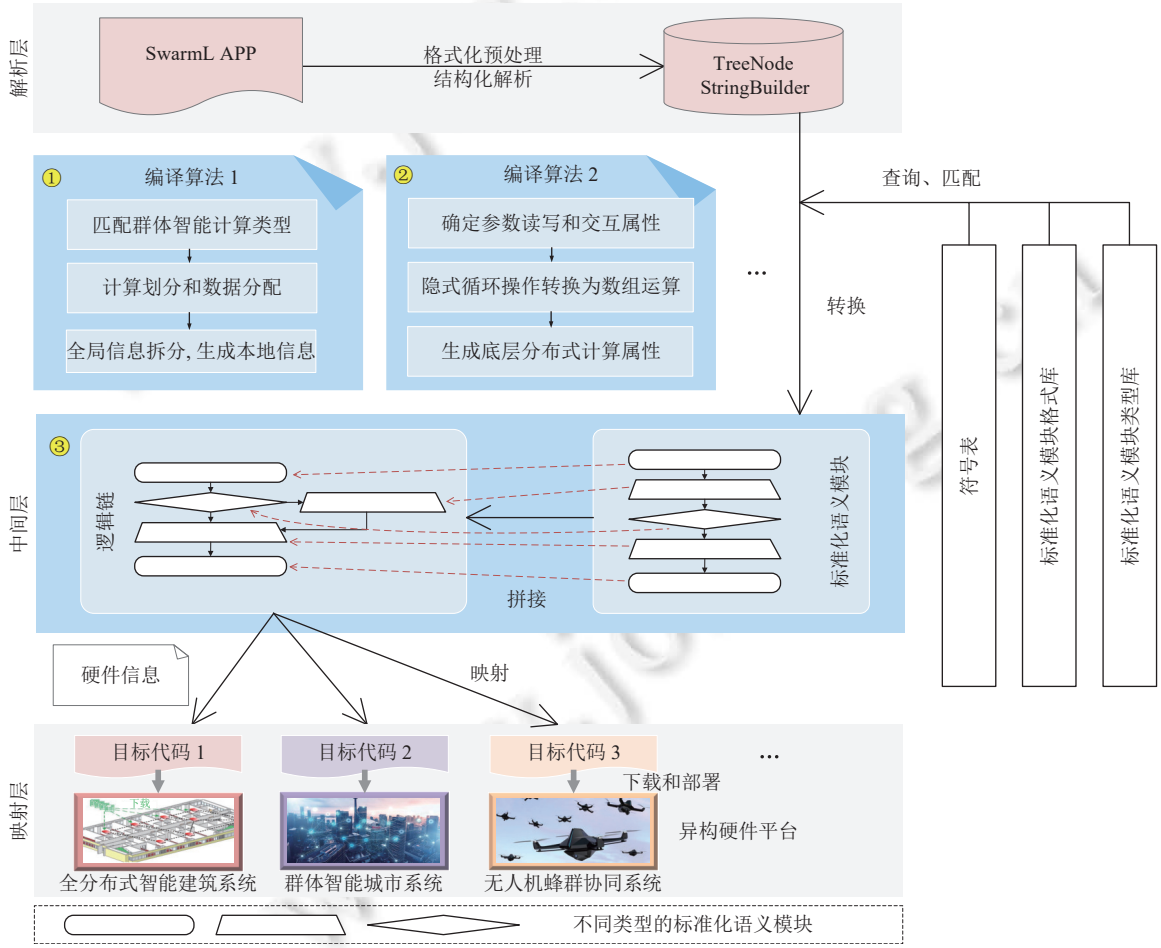


图 4 面向群体智能系统应用程序的编译系统总体框架

© 中国科学院软件研究所 http://www.jos.org.cn

解析层负责格式化和预处理 SwarmL 程序, 对源程序进行词法语法分析, 建立符号表, 生成抽象语法树并存储在 StringBuilder 数据结构中.

针对挑战 1, 本文提出一种编译算法, 将符合用户串行思维方式、面向全局角度编程的群体智能系统应用程序解耦、分离为面向单个节点独立执行的中间代码. 首先根据应用程序中全局参数、计算域和操作符等元素匹配生成群体智能计算的类型. 然后识别并提取串行应用程序中的全局信息进行计算划分和数据分配, 将全局信息转换为每个节点需要在本地定义以及与邻居交互的信息. 确定发起点和参与节点的计算逻辑, 使得每个节点能够根据自身属性来自动识别自身所要执行的程序代码块并与邻居交互, 实现全局层面的群体智能计算任务.

针对挑战 2, 本文提出一种编译算法, 考虑到从邻域角度编程的 Swarm 并行语句与 C 语言编程方法的相似性, 将声明式、面向邻域角度编程的 Swarm 并行语句转换为面向单个节点独立执行的 C 语言程序并嵌入到中间代码中, 进而重用 C 语言编译器并降低编译成本. 首先识别并提取 Swarm 并行语句中的并行信息, 根据局部参数的计算域判断参数的本地读写和邻域交互属性, 并生成参数对应的 C 语言变量和数组. 将 Swarm 并行语句中对邻居的隐式循环遍历操作转换为基于 C 语言数组的运算. 根据参数的计算域生成计算触发条件、计算结果发布条件等分布式计算属性, 以支持节点与邻居的并行交互计算. 将生成的 C 语言程序嵌入到中间代码中, 进而重用 C 语言编译器并降低编译成本.

利用上述 2 个核心编译算法遍历抽象语法树, 通过查询符号表以及预先设置的标准化语义模块格式库和类型库, 生成由多种类型语义模块组成的标准化语义模块序列.

针对挑战 3, 本文设计了一种标准化的编译系统中间表示. 在分析群体智能系统体系结构和程序执行模型的基础上, 该中间表示将每种类型的群体智能运算表示为输入输出变量和标准化群体智能算子相结合的计算事件, 本文称之为标准化语义模块. 因而具有并行和分布式特征的群体智能应用程序被转换为标准化语义模块按照语义执行顺序拼接而成的序列, 本文称之为逻辑链. 标准化语义模块和逻辑链组成编译系统中间表示, 这种通用化的中间表示将群体智能计算类型和数据有效解耦, 能够适配一类具有节点交互协作、并行执行特征的群体智能系统硬件平台, 进而能够解决分别编写程序代码和编译的困难, 屏蔽底层硬件平台的异构性. 同时, 该中间表示为群体智能应用程序和底层硬件平台提供统一的标准化程序接口, 当两者任一发生更新升级时, 只需修改中间表示的接口而无需修改整个编译系统软件, 进而大大降低了编译系统维护的工作量.

在中间层, 根据 SwarmL 程序的实际语义执行顺序, 将原本按照语法顺序排列的标准化语义模块序列拼接成具有分支、跳转等关系的逻辑链. 逻辑链与上层应用程序所描述的算法逻辑一致, 足够灵活, 且接近硬件.

在映射层, 通过修改接口格式和输入硬件平台信息, 逻辑链能够被映射为在不同类型的群体智能系统平台上运行的目标代码, 例如全分布式智能建筑系统, 基于群体智能的智慧城市系统, 无人机蜂群协同系统等, 如图 4 所示. 本文使用扁平化、无中心的全分布式智能建筑系统作为群体智能系统硬件平台的示例来说明从逻辑链到目标代码的映射转换过程.

2.2 编译系统中间层格式

本节定义编译系统中间层接口的标准化格式以及标准化语义模块类型库, 其对应于第 1.4 节的编译挑战 3. 中间层对于后期代码生成至关重要, 因为高级 SwarmL 程序隐藏了大量细节, 这使得很难直接从 SwarmL 生成面向底层硬件平台的目标代码.

2.2.1 定义编译系统中间层接口格式

编译系统中间层包括标准化语义模块和逻辑链.

(1) 标准化语义模块

在分析群体智能系统体系结构和程序执行模型的基础上, 编译系统中间层将每个群体智能操作表示为输入输出变量和标准化群体智能算子相结合的计算事件, 即标准化语义模块. 其可以表示为一个四元组.

$$\text{Standardized Semantic Module} = <M_{Id}, M_{Name}, M_{Type}, Var_{In}, Var_{Out}>,$$

其中,

© 中国科学院软件研究所 http://www.jos.org.cn

● M_{Id}: 语义模块的序号, 由编译系统的解析层在生成每一个标准化语义模块时给出, 在上一个语义模块序号的基础上递增. 初始语义模块序号为 00000001.

● M_{Name}: 语义模块的名称, 对应于 SwarmL 程序中场变量定义、表达式、语句等不同类型语言要素的名称.

● M_{Type}: 语义模块的类型码. SwarmL 中场变量定义、表达式、语句等不同类型的语言要素都被归纳为若干种类的语义模块. 类型码表示每个语义模块所执行的群体智能计算逻辑.

● Var_{In}, Var_{Out}: 输入和输出变量名称. 每种语义模块都对应相应的参与计算的输入变量, 和计算结果输出变量. 不同类型的语义模块, 因其计算功能不同, 其输入输出变量的个数和描述方法可能不同.

(2) 逻辑链

语义模块序列仅按照 SwarmL 程序的语法顺序对语义模块进行了简单排列, 并不能反映实际语义执行顺序. 将语义模块看作基本单元, 根据每一个语义模块的后续模块的名称以及执行这些后续模块的逻辑条件, 就可以完整描述整个逻辑链, 其反映了全分布式智能建筑系统控制任务的整体逻辑. 从逻辑链中的某个语义模块的角度, 逻辑链可以表示为一个四元组:

$$\text{Logic Chain} = <M_{Id}, M_{NextNameList}, Var_{Logic}, Cond_{Logic}>.$$

● M_{Id}: 本语义模块的序号, 是该模块在整个逻辑链中唯一的标识.

● $M_{NextNameList}$: 本语义模块的后续 1–n 个语义模块的名称. 每个语义模块可以有多个后续语义模块, 也可以没有后续语义模块.

● Var_{Logic}: 到后续 1–n 个语义模块的逻辑判断变量名称, 也是模块间同步并行执行的依据. 本语义模块后面跟随的各个模块, 可以有条件执行或者无条件执行. 逻辑链中, 允许用某个变量的值作为判定条件来判定语义模块是否执行.

● $Cond_{Logic}$: 逻辑条件. 当后续语义模块的执行有条件时, 本字段描述后续模块的执行, 是在对应判断变量的逻辑值为 1 时执行, 还是在逻辑值为 0 时执行. 例如, 当逻辑判断变量的值为 0 时, 执行语义模块序号为 M1 的后续模块, 当该变量的值为 1 时, 执行语义模块序号为 M2 的后续模块.

中间层既携带高级的群体智能应用程序中的计算信息, 又便于之后的代码生成过程. 在后续代码生成阶段, 通过语义模块中包含的群体智能算子可以识别每个语义模块的功能, 并生成相应的目标代码. SwarmL 程序和标准化语义模块、逻辑链的对应关系如后文图 5 所示, 其中每条语句对应一个标准化语义模块. 具体的中间代码格式文档和案例程序已开源[32].

中间层将每种类型的群体智能运算表示为输入输出变量和群体智能算子相结合的通用化语义模块序列, 实现了群体智能计算类型和数据的解耦, 并且其标准化定义了语义模块之间执行的先后顺序关系. 标准化语义模块和其中的群体智能算子对 SwarmL 语言进行了一定封装, 在编译过程中无需进行全局再编译. 只需修改中间层中输入输出变量接口以及中间层到目标代码的接口规则, 该中间层即可应用至不同类型的群体智能硬件平台, 进而屏蔽目标硬件平台的异构性. 这种将领域特定语言转换为标准化中间表示进而适应不同硬件平台的思想方法, 具备一定的普适性, 对其他领域特定语言的编译开发具有借鉴意义.

2.2.2 构建标准化语义模块类型库

SwarmL 程序中场变量、计算域等元素代表着不同类型的群体智能计算逻辑, 但这些高级编程抽象无法直接被硬件平台所识别. 本节根据源程序中不同语言元素的特征建立对应的标准化语义模块类型库. 通过说明每种语义模块对应的 SwarmL 元素的表达形式, 为标准化语义模块序列的生成提供模板信息.

以全局参数表达式为例, 运算符的差异和两侧参数的计算域差异等都可能产生不同的并行计算语义, 进而产生不同类型的语义模块. 例如, 对于第 1.3 节的案例 1 中的表达式, 当等号左侧参数的计算域为 All, 右侧参数计算域为 StartNode 时能够实现全局扩散任务, 而当等号右侧为常数 constant 时, 表示所有节点的本地运算. 因此, 从运算符和计算域两个维度出发, 排列组合两个维度中的不同特征, 可以建立对应于全局参数表达式的标准化语义模块类型库.

表 1 以赋值、加法和乘法运算表达式为例, 展示了左右两侧参数的计算域的组合所对应的语义模块类型. 完

© 中国科学院软件研究所 http://www.jos.org.cn

整的标准化语义模块类型库已开源[32]. 编译系统在遍历 SwarmL 程序的抽象语法树时, 根据运算符、计算域等元素查询该语义模块类型库, 能够匹配并生成该程序段所对应的标准化语义模块类型. 图 5 给出了 SwarmL 程序与标准化语义模块类型库的对应关系.

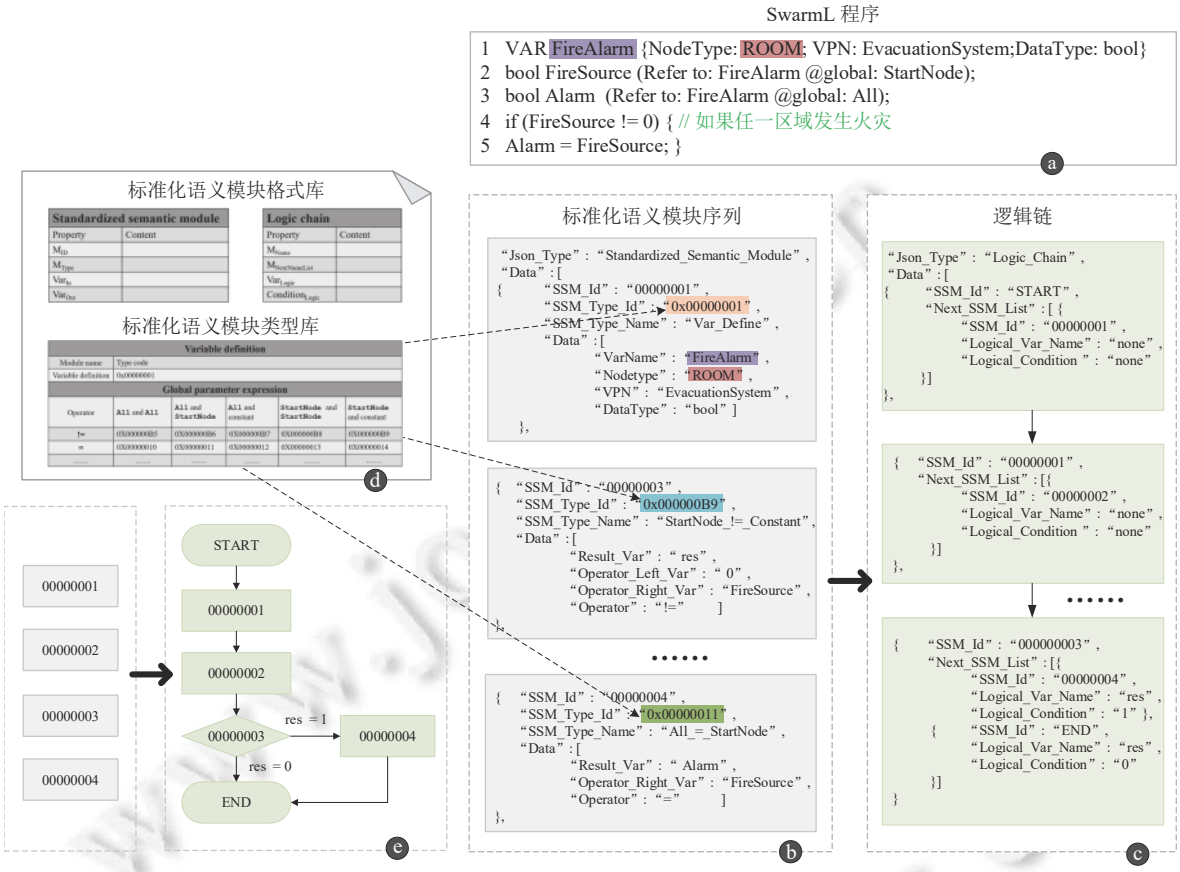


图 5 SwarmL 源程序与标准化语义模块和逻辑链对应示意图

表 1 全局参数表达式对应的标准化语义模块类型库

运算符	运算符左右两侧参数的计算域				
	All 和 All	All 和StartNode	All 和 constant	StartNode 和 StartNode	StartNode 和constant
=	0x00000010	0x00000011	0x00000012	0x00000013	0x00000014
+	0x00000020	0x00000021	0x00000022	0x00000023	0x00000024
*	0x00000040	0x00000041	0x00000042	0x00000043	0x00000044

3 编译系统实现

本节实现了面向群体智能系统应用程序的编译系统. 提出编译群体智能系统应用程序的具体算法, 包括源程序预处理和结构化解析, 源程序到标准化语义模块序列, 语义模块序列到逻辑链, 以及逻辑链到目标平台指令程序的编译算法. 另外, 开发了具有程序编辑和目标代码生成功能的可视化编译系统工具.

3.1 源程序预处理和结构化解析

该部分对源程序进行结构化解析并分类存储以为后续编译阶段提供输入. 如图 6 所示, 首先格式化处理 SwarmL 程序, 以删除冗余的空格字符, 并合并或分割行, 避免编译期间的歧义. 根据 SwarmL 中场变量定义、表达

© 中国科学院软件研究所 http://www.jos.org.cn

式等元素的正则表达式规则, 对源程序进行结构化解析, 提取程序信息, 并将代码分割成逻辑块, 处理块中的每一行. 每类 SwarmL 元素被拆分成携带语法信息的符号流.

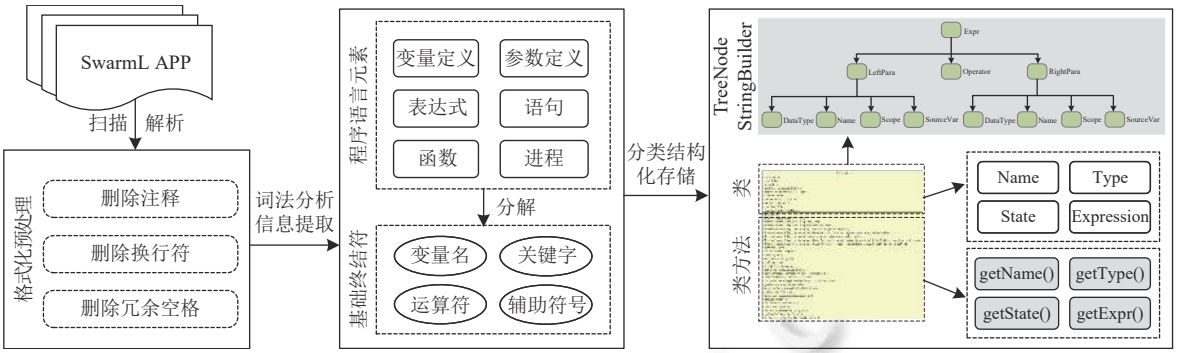


图 6 源程序预处理和结构化解析流程

为了给源程序解析后的存储提供统一的数据结构和操作, 本节定义了包含 TreeNode 类的 StringBuilder 数据结构. 其中 TreeNode 类主要由 Name、Type、Expression 等属性参数组成, 负责结构化存储 SwarmL 元素的各项属性信息, 例如场变量的名称, 数据类型等. 提供了 getName、setName、getType 等类方法来实现对属性参数的访问和操作. 然后, 将源程序字符串集合组合成抽象语法树存储在 StringBuilder 数据结构中. 在后续编译过程中通过查询访问语法树结点, 生成标准化语义模块的相关信息. 以全局参数表达式的解析为例, 提取出其运算符以及左右两侧参数的信息并存储于 StringBuilder 数据结构中. 在图 6 右侧的抽象语法树中, Expr 是该表达式对应的语法树结点类型, 访问该结点可以获取运算符, 左右参数的名称, 数据类型, 计算域, 参数所指向的场变量等信息.

3.2 源程序至标准化语义模块序列编译算法

针对第 1.4 节所分析的编译挑战, 本节提出将符合用户串行思维、面向系统全局编程的场变量和全局参数表达式编译为面向单个节点执行的标准化语义模块序列的编译算法, 以及将声明式、面向邻域编程的 Swarm 并行语句转换为 C 语言代码的编译算法.

3.2.1 全局串行的场变量定义语句编译算法

场变量支持显式刻画分布式建筑物理场, 允许用户以全局声明式的编程范式在域中符合条件的所有节点上直接定义变量, 而无需考虑节点之间的任务分配和数据交互. 而在程序运行时, 该计算任务需要根据场变量对应的节点类型和域来创建虚拟计算网络, 该网络在域 VPN 中, 由节点类型为 NodeType 的节点组成. 网络中触发该场变量定义计算任务的节点被确定为发起点, 以发起点作为根节点逐级建立通信生成树 [2]. 在发起点定义临时变量, 并赋初值给新定义的变量. 然后将被定义变量的名称、分布属性等字段发送给下游节点, 网络中所有节点都按照此逻辑执行. 最后, 所有节点都根据发起点传来的若干被定义变量的名称、分布属性等字段, 在本节点定义场变量在本地投影的变量.

由于场变量定义语句 (FVD) 和标准化语义模块 (SSM) 元素之间的匹配关系较为复杂, 本文采用模型管理方法 [33] 形式化定义两者之间的映射关系, 其具有足够的语义表现力和处理能力, 如图 7(a) 所示. 场变量定义中的关键字被映射为语义模块类型和名称. 场变量名称, 节点类型集合, 域名称和数据类型都被映射为语义模块的输出变量. 如下所示.

FVD = <Keyword, VarName, NodeType, VpnName, DataType>;

$SSM = <M_{Id}, M_{Name}, M_{Type}, Var_{In}, Var_{Out}>$;

$SSM.M_{Id} = Pre(SSM.M_{Id}) + 1$;

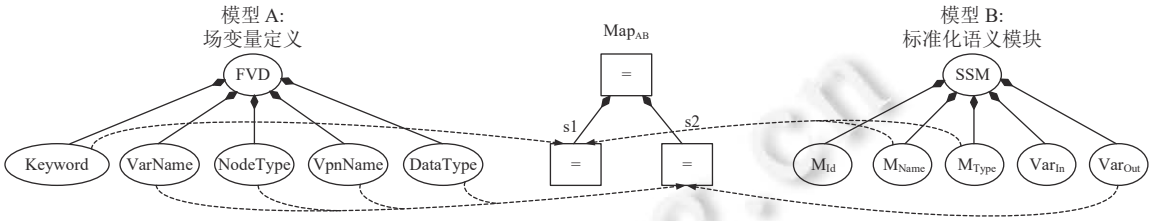
$FVD.Keyword = M.s1, M.s1 = (SSM.M_{Name}, SSM.M_{Type})$;

$(FVD.VarName, FVD.NodeType, FVD.VpnName, FVD.DataType) = M.s2, M.s2 = SSM.Var_{Out}$.

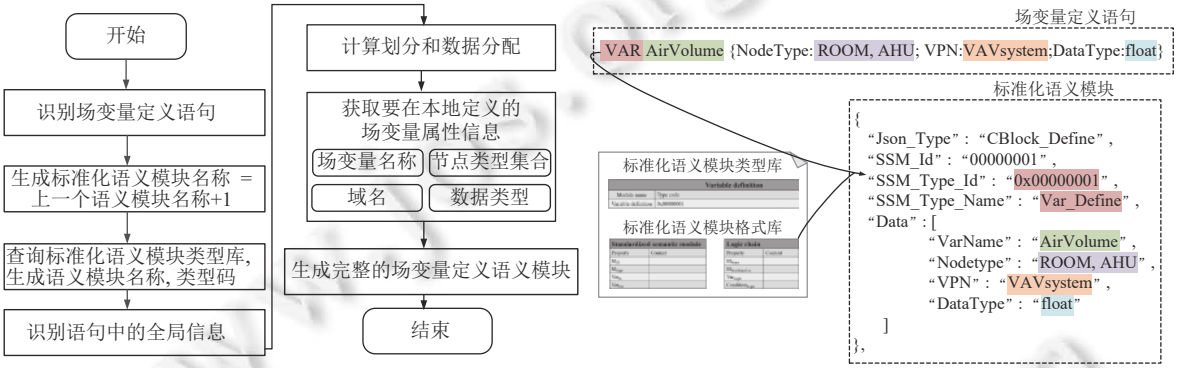
本节提出将符合用户串行思维、面向全局编程的场变量定义语句转换为标准化语义模块的编译算法, 其旨在解决第 1.4 节的编译挑战 1. 算法流程如图 7(b) 所示, 当编译系统根据正则表达式检测到源程序段是场变量定义

© 中国科学院软件研究所 http://www.jos.org.cn

语句时, 首先生成一个其对应的语义模块, 该语义模块的编号 M_{id} 在上一个生成的语义模块的基础上递增 1, 初始语义模块编号为 00000001. 遍历其抽象语法树, 识别场变量定义语句的关键字, 通过查询标准化语义模块类型库, 匹配生成语义模块的名称 M_{Name} 为 Var_Define, 语义模块类型码 M_{Type} 为 0x000001. 识别场变量定义语句中的全局信息, 进行计算划分和数据分配, 获取每个节点要在本地定义的场变量属性信息, 包括场变量的名称, 数据类型, 对应的节点类型集合和域名, 并映射生成标准化语义模块的输出变量. 图 7(c) 给出了场变量定义语句到标准化语义模块的编译转换示例.



(a) 场变量定义语句与标准化语义模块的映射关系



(b) 场变量定义语句编译算法流程　　(c) 场变量定义语句编译转换案例

图 7　场变量定义语句编译示意图

该算法的关键特征在于: 场变量定义语句在网络全局层面进行计算描述, 而 SwarmL 编译系统基于通信生成树机制, 将对群体任务的全局描述转换为个体节点的控制行为.

3.2.2　全局参数表达式编译算法

全局参数是场变量在计算域上的数据投影, 其表示场变量在发起点或者域中所有节点上的数据集合. 全局参数表达式通过组合不同参数、计算域和运算符等并行原语, 直观刻画了全局扩散、求和等复杂的群体智能计算任务, 而无需考虑通信和数据分配等底层计算细节. 以第 1.3 节中的扩散程序为例, 在程序运行时, 构建以发起点为根节点的生成树, 从发起点开始自根部到末端依次执行. 各节点都将发起点传来的参数数值与本节点上的参数数值进行运算, 将结果赋值给等号左边的本地参数, 进而实现火灾报警信号全局扩散的效果.

图 8(a) 显示了全局参数表达式 (GPSta) 和标准化语义模块 (SSM) 元素间的映射关系. 根据表达式中的运算符和左右两侧参数的计算域匹配查询标准化语义模块类型库, 映射生成语义模块的类型和名称. 运算符左右两侧的参数名称被映射为语义模块的输入变量. 表达式运算的结果变量被映射为语义模块的输出变量.

GPSta = <LeftGlobalParameter, Operator, RightGlobalParameter, ResultVar>;

SSM = <M_{Id}, M_{Name}, M_{Type}, Var_{In}, Var_{Out}>;

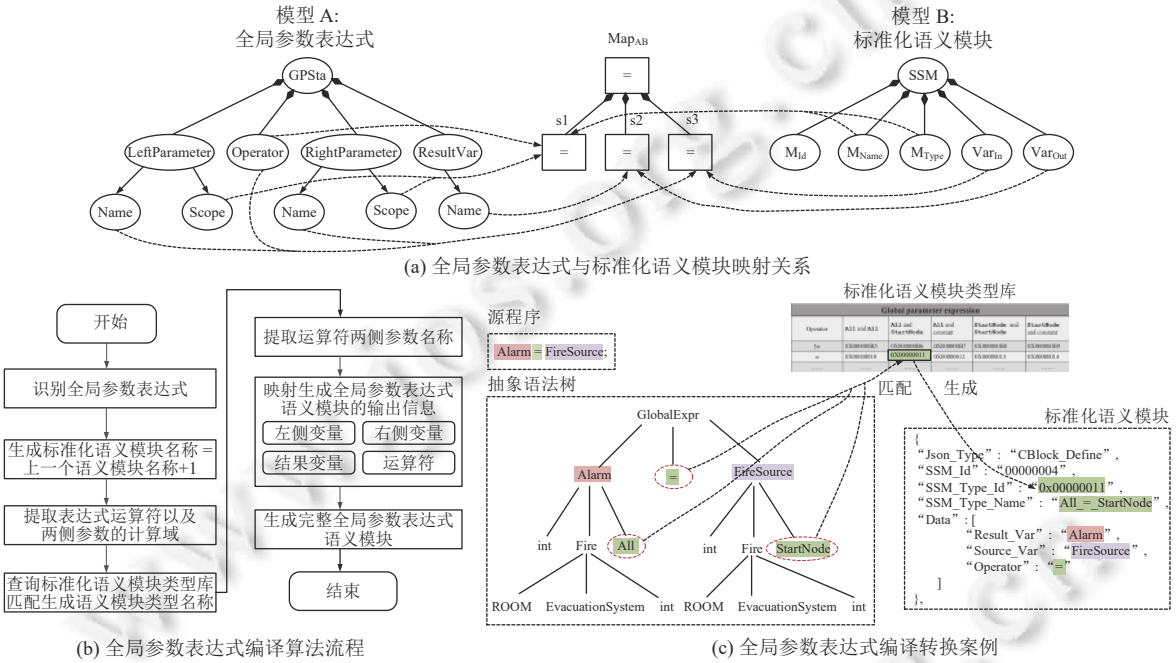
$SSM.M_{Id}$ = Pre($SSM.M_{Id}$) +1;

(GPSta.LeftScope, GPSta.Operator, GPSta.RightScope) = M.s1, M.s1= ($SSM.M_{Name}$, $SSM.M_{Type}$);

© 中国科学院软件研究所 http://www.jos.org.cn

(GPSta.LeftName, GPSta.Operator, GPSta.RightName) = M.s2, M.s2 = SSM.Var_{In};
GPSta.ResultVarName = M.s3, M.s3 = SSM.Var_{Out}.

本节提出将符合用户串行思维的全局参数表达式转换为标准化语义模块的编译算法, 其旨在解决第 1.4 节的编译挑战 1. 算法流程如图 8(b) 所示, 当识别出源程序段是全局参数表达式时, 生成语义模块的编号在上一个模块的基础上递增 1. 遍历表达式抽象语法树, 根据表达式中的运算符和左右两侧参数的计算域匹配查询标准化语义模块类型库, 判断表达式对应的群体智能计算类型, 进而映射生成语义模块类型码 M_{Type} 和名称 M_{Name}. 识别表达式中的全局数据信息, 提取运算符两侧参数的名称, 生成语义模块的输入变量. 提取代表表达式运算结果的参数名称, 生成语义模块的输出变量. 图 8(c) 给出全局参数表达式到标准化语义模块的编译转换示例.



(a) 全局参数表达式与标准化语义模块映射关系

(b) 全局参数表达式编译算法流程

(c) 全局参数表达式编译转换案例

图 8 全局参数表达式编译示意图

该算法的关键特征在于: 根据表达式中操作符和全局参数的计算域的组合, 判断该表达式所要执行的群体智能运算类型, 并生成单个节点执行层面的代码. 上述转换逻辑仅针对包含单个运算符的简单全局参数表达式. 由于每个表达式标准化语义模块只能描述单个简单表达式的逻辑, 当处理由多个简单表达式组成的复杂表达式时, 需要按照运算符优先级顺序将其转换为由若干个简单表达式语义模块组成的序列.

3.2.3 面向邻域计算的 Swarm 并行语句编译算法

与全局参数表达式所采用的全局编程视角不同, Swarm 并行语句使用节点本地及其邻居节点的信息描述群体智能计算逻辑, 因而语句中出现的都是局部参数. 当描述邻域范围的计算逻辑时, SwarmL 与通用 C 语言的语法相近. 为了利用现有资源并降低编译难度, 将 Swarm 并行语句转换为 C 语言程序并嵌入标准化语义模块, 以便于直接调用 C 语言编译器将 C 程序编译为二进制代码并将其链接到目标代码中.

Swarm 并行语句支持以局部参数声明式的描述对多个邻居数据的读写操作, 且无需考虑邻居间参数的读写和交互机制, 以及计算任务如何触发、计算结果发布给哪些邻居等底层分布式计算属性. 但 C 语言面向节点独立执行阶段, 需要显式描述参数的读写机制等底层计算细节. 因此, Swarm 并行语句到 C 语言程序转换需要还原 SwarmL 程序所涉及的分布式计算的底层细节. 本节提出将面向邻域并行计算的 Swarm 并行语句转换为标准化语义模块编译算法, 其旨在解决第 1.4 节的编译挑战 2. 该算法主要包括生成局部参数对应的 C 语言变量, 将对邻居

© 中国科学院软件研究所 http://www.jos.org.cn

的隐式遍历操作转换为基于 C 语言数组的运算, 以及生成 Swarm 并行语句所涉及的分布式计算属性 3 个部分, 编译案例如图 9 所示. 由于篇幅限制, Swarm 并行语句具体编译算法由附录中图 A2 给出.

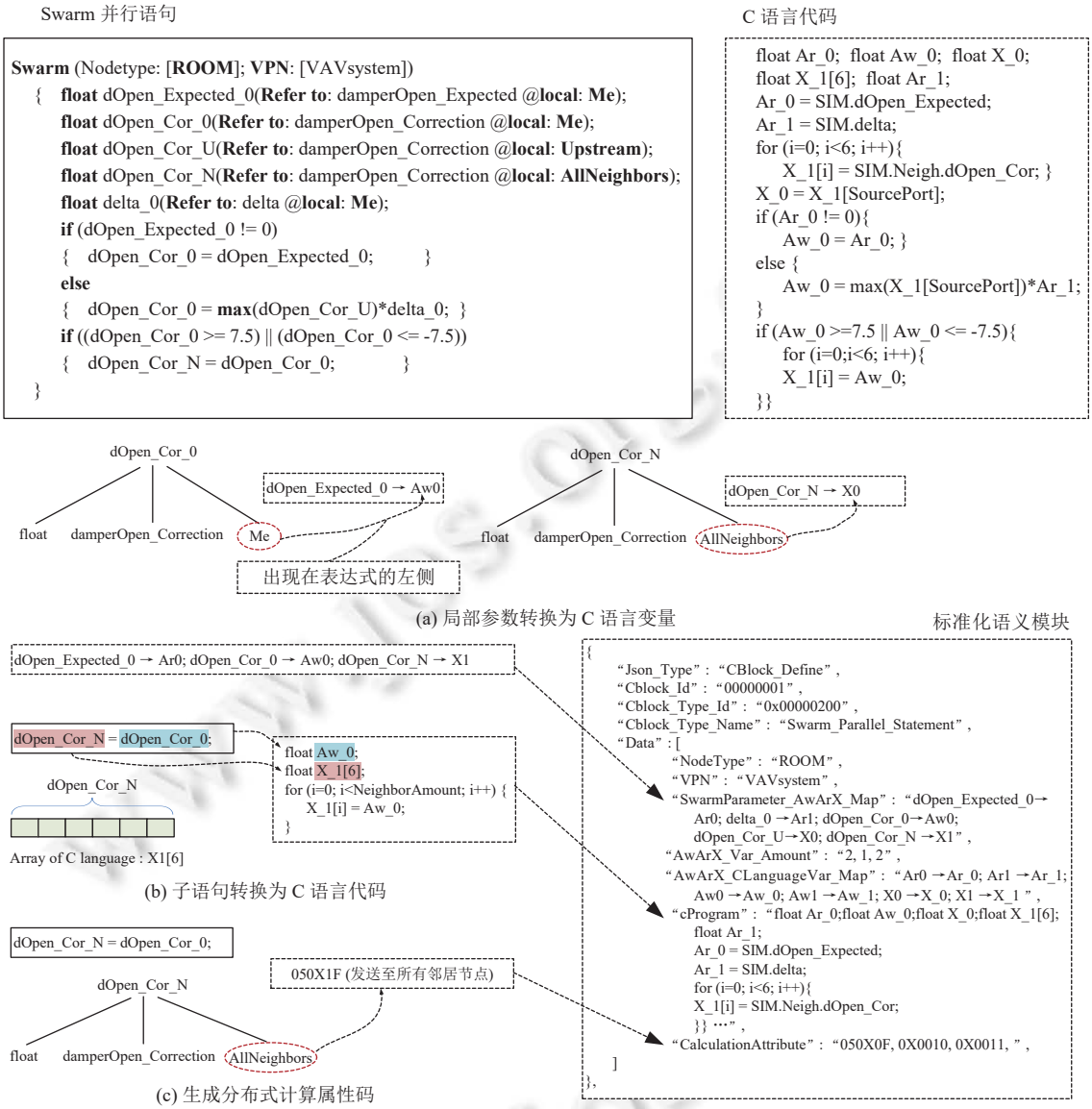


图 9 Swarm 并行语句编译示意图

(1) 局部参数转换生成 C 语言中对应的本地读, 本地写和交互变量

Swarm 并行语句中局部参数的计算域属性隐式表达了参数的读写机制, 屏蔽了分布式计算中消息发送和接收等通信细节. 但在 C 语言程序中要求显式描述变量的读写机制, 因此需要将 Swarm 并行语句中的参数转换为 C 语言中相应的变量. 本编译系统规定在计算逻辑中只读的本地参数对应的 C 语言变量称为本地输入变量, 即 Ar 变量; 将可以修改的本地参数对应的变量称为本地输出变量, 即 Aw 变量; 将读取或写入邻居节点参数对应的变量称为交互变量, 即 X 变量. 例如, 计算域为 AllNeighbors 的局部参数 dOpen_Cor_N 表示站在本节点角度所观察到的所有邻居节点上的风阀开度校核值, 对应于交互变量 X.

该部分算法逻辑为, 遍历 Swarm 并行语句抽象语法树中的所有局部参数, 根据参数名称查询该参数的计算域

© 中国科学院软件研究所 http://www.jos.org.cn

属性, 并判断该参数在表达式中出现的位置, 进而匹配生成参数对应的 Ar, Aw, X 变量 (详细对应关系在文献 [6] 中第 4.3 节介绍). 最后, 生成 Ar, Aw, X 变量序列以及参数与 Ar, Aw, X 变量的映射关系表, 如图 9(a) 所示.

(2) 子语句转换为 C 语言代码

具有邻域计算域属性的局部参数表示该参数在邻域中某些节点上的数据集合. 例如, 计算域为 AllNeighbors 的参数 dOpen_Cor_N. Swarm 并行语句对所有邻居的循环、遍历操作是隐式的, 这种描述方式虽然简洁直观, 但在通用化的 C 语言中是不允许的. C 语言支持以数组形式表示多个邻居的数据, 并通过数组运算实现对邻居数据的显式循环、遍历操作. 因此, 需要将 Swarm 并行语句的子语句转换为基于数组运算的 C 语言程序.

该部分算法逻辑为, 将对应于 X 变量的参数映射为 C 语言数组, 并生成基于数组运算的 C 语言代码. 以子语句 dOpen_Cor_N = dOpen_Cor_0 为例. dOpen_Cor_0 对应于本地写变量, dOpen_Cor_N 对应于交互变量. 因此, 当解析该语句时, 首先将 dOpen_Cor_0 转换生成 Aw_0 变量, 将 dOpen_Cor_N 转换生成 X1[i] 数组. 构造 for 循环语句, 生成 i 为新定义的 C 语言临时变量, 用于记录 for 循环次数, 获取参数 dOpen_Cor_N 的计算域中包含的节点数目并设置为循环增量上限, 每循环一次都用 X1[i] 来替换参数 dOpen_Cor_N 在 Swarm 并行语句中的位置, 其中 X1[i] 表示参数中对应第 i 个邻居的数值, 进而转换生成该子语句对应的 C 语言代码, 如图 9(b) 所示.

(3) 根据参数的计算域生成分布式计算属性码

节点计算如何触发、计算结果发布对象等分布式计算属性在 Swarm 并行语句中以参数计算域的形式隐式表示. 因此, 需要根据 Swarm 并行语句中参数的计算域匹配生成各项计算属性对应的属性码. 以计算结果发布对象属性为例, 其描述的是单个节点单次计算的结果需要发送给哪些邻居节点.

计算结果发布对象属性的生成算法如下: 当 Swarm 并行语句中不包含对应于 X 变量的参数时, 表示本次计算不会将计算结果告诉任何邻居节点, 生成属性码为 050x00. 当语句中对所有 X 变量的最后一次赋值, 等式左侧 X 变量对应的参数计算域为 AllNeighbors 时, 表示将计算结果发布给所有邻居, 生成属性码为 050x0F, 如图 9(c) 所示.

综上, 按照步骤 (1)–(3), Swarm 并行语句被编译为标准化语义模块序列.

3.3 标准化语义模块至逻辑链编译算法

在编译系统中间层, 标准化语义模块序列被按照分支、跳转、循环等语义逻辑组合成与 SwarmL 程序实际执行顺序一致的逻辑链, 具体实现方法如算法 1 所描述. 以每一个语义模块为基本单元, 首先生成每个模块的编号 SSM_{ID}. 然后, 为每个语义模块设置逻辑判断变量, 其对应的是源程序分支、跳转、循环语句中承载条件表达式运算结果的临时变量. 当语义模块不是条件表达式时 (例如场变量定义和赋值语句等), 表示顺序执行关系, 逻辑判断变量 Logical_Var 和逻辑判断条件 Logical_Condition 都为 none, 后续语义模块编号即为本语义模块的编号加 1. 当语义模块是条件表达式时, 表示存在分支、跳转和循环关系, 根据逻辑判断变量取值为 0 或者 1 来生成后续语义模块编号. 当逻辑判断变量为 1 时, 本语义模块的后续模块的编号即为满足条件后执行的语义模块编号. 反之, 当逻辑判断变量为 0 时, 本语义模块的后续模块的编号即为不满足条件时所执行的语义模块编号. 图 5 给出了标准化语义模块到逻辑链的编译过程示例.

该算法能够确定每个语义模块在实际执行时的后续模块的编号, 进而生成了以语义模块为基本单位且具有完整、直观执行顺序的逻辑链. 逻辑链将全分布式智能建筑系统控制任务描述为若干计算事件经过排列组合而成的序列, 其与 SwarmL 语言所描述的算法逻辑相一致, 足够灵活且接近硬件, 可以生成硬件平台可执行的目标指令程序.

算法 1. 标准化语义模块至逻辑链编译算法.

输入: 标准化语义模块序列;
输出: 逻辑链.

1. **for** each standardized semantic module
2. Generate SSM_{ID}
3. // 生成后续语义模块列表
4. **if** $O_p \neq \forall(<, \leqslant, >, \geqslant, !=, ==)$ **then** //如果不是条件表达式

© 中国科学院软件研究所 http://www.jos.org.cn

5. $SSM_{NextID} = SSM_{ID} + 1$
6. Logical_Var ← “none”, Logical_Condition ← “none”
7. **else**
8. Logical_Var ← OutputVar
9. Logical_Condition ← “1”
10. SSM_{NextID} = ID of the next module to execute when the condition is met
11. Logical_Var ← OutputVar
12. Logical_Condition ← “0”
13. SSM_{NextID}= ID of the next module to execute when the condition is not met
14. **end if**
15. **end for**

3.4 底层指令序列生成算法

为了生成面向异构平台的可执行目标代码, 需根据不同的平台特征和转换规则, 对逻辑链进行进一步变换和处理, 生成平台可执行的目标代码. 本文以全分布式智能建筑系统平台[2]作为群体智能系统的一个典型例子来说明逻辑链到目标指令程序的转换. 全分布式智能建筑系统提供了面向智能节点操作系统[27]的底层指令序列 (UIS) 来支持应用程序的运行时. 作为目标代码, 底层指令序列包含变量定义、赋值、查询、函数定义等基础操作. 建筑运行控制任务被看作是按照一定顺序执行的底层指令的序列. 由于生成的语义模块是标准化的, 逻辑链和底层指令序列之间可以建立映射关系. 可以为逻辑链中的每一种语义模块配置相应的底层指令, 可能是某一个指令, 也可能是一系列指令构成的标准化组合. 具体实现方法如算法 2 所描述.

算法 2. 逻辑链到底层指令序列编译算法.

输入: 逻辑链;
输出: 底层指令序列.

1. **for** each standardized semantic module in the logic chain
2. $SSM_{Type\_ID} \rightarrow UIS_{Type\_ID}$
3. **if** field-oriented variable definition semantic module **then**
4. SSM ($VarField_{Name}$, $VarField_{DataType}$) → UIS (Var_{Name}, Var_{Bytes}, $Var_{DataType}$, Var_{Number})
5. **else if** Swarm parallel statement semantic module **then**
6. SSM (NodeType, VPN) → UIS (Scope, hop count)
7. SSM (Computing attribute code) → UIS (Computing attribute set)
8. $Table_{Map}$ ($SSM_{Parameter}$ → (Ar, Aw, X)), $Table_{Map}$ ((Ar, Aw, X) → C language variables) →UIS (Name, number, data type, and bytes of Ar, Aw, X variables)
9. SSM (Generated C language code)→UIS (content, verification results, and total length of C code)
10. **else if** other SwarmL elements semantic module **then** …
11. **end if**
12. **end for**
13. Arrange and generate all instruction sequences in the order shown in the logic chain

首先, 遍历逻辑链中的每个语义模块, 根据语义模块的类型和编号生成对应的底层指令类型和编号. 然后, 逐个将语义模块中的其他信息映射至底层指令中的信息. 以 Swarm 并行语句语义模块为例, 首先将语义模块类型映射至底层指令序列类型, 根据语义模块中的节点类型和 VPN 映射生成底层指令中的作用域和跳数, 根据分布式计

© 中国科学院软件研究所 http://www.jos.org.cn

算属性码的集合数组映射生成底层指令中的计算属性, 根据局部参数信息和 Ar, Aw, X 变量到 C 语言变量的映射关系生成底层指令中 Ar, Aw, X 变量的个数, 名称, 数据类型和字节长度等信息, 根据第 3.2.3 节中生成的 C 代码映射生成指令中所需要的 C 代码的总长度, 校验结果和内容等.

其他类型语言要素对应语义模块的转换算法与此类似. 最后, 按照逻辑链中所包含的先后执行顺序关系, 将这些分段底层指令序列链接形成完整的底层指令序列, 即智能节点可执行的应用程序. 至此, 编译系统的解析层、中间层和转换层逐步将 SwarmL 程序编译转换为全分布式智能建筑系统平台可执行目标代码.

另一方面, 许多群体智能系统编程方法具有与 SwarmL 语言类似的程序元素和编程模式. 例如, 物联网编程模型 PatRICIA[34]定义了 IntentScope 原语来动态划分一组物理设备, 机器人群编程语言 Buzz[7]和无人机群编程框架 PaROS[35]定义了 Swarm 和 neighbor 原语来对不同范围的群体执行可扩展的集合操作. 这些编程原语虽然种类不同, 但都支持动态指定群体智能运算任务的计算类型和分布数据. 其与本文标准化中间层中的群体智能算子和计算域具有相似功能, 都能够方便用户简洁编程群体智能系统应用程序. 因此, 该中间层具有一定的普适性, 通过建立 SwarmL 群体智能算子到其他语言原语的映射关系, 修改中间层输入输出变量接口以及中间层到目标代码的接口规则, 可以生成面向机器人群等不同类型群体智能硬件平台的可执行目标代码.

3.5 编译系统工具开发

本文基于 Xtext 插件[36]开发了面向群体智能系统应用程序的可视化编译系统工具. 图 10 给出该编译系统工具的基本架构, 其主要包括源程序结构化解析器和代码生成器两部分. 利用 Java 开发的源程序结构化解析器 (Parser.java) 包括源程序扫描器, 格式化器, 以及解析器. 其中, 源程序格式化器负责获取 SwarmL 程序代码源路径并读取程序内容, 将 SwarmL 程序格式化, 即删除注释、标点符号、空白区域和换行符等内容. 源程序扫描器逐行读取关键字、函数、场变量、全局参数表达式等程序内容, 生成携带语法信息的符号流. 解析器根据正则表达式将 SwarmL 元素解析为抽象语法树, 并存储在 TreeNode 中. TreeNode 包括结点名称, 类型, 状态等类以及调用这些类的方法. 代码生成器遍历源程序抽象语法树 TreeNode, 根据所设计的标准化语义模块格式库和语义模块类型库以及 SwarmL 具体编译算法, 生成标准化语义模块序列. 然后将语义模块序列按照实际执行顺序转换为逻辑链, 最后设置目标代码生成路径及格式, 将逻辑链映射为面向全分布式智能建筑系统平台的目标代码.

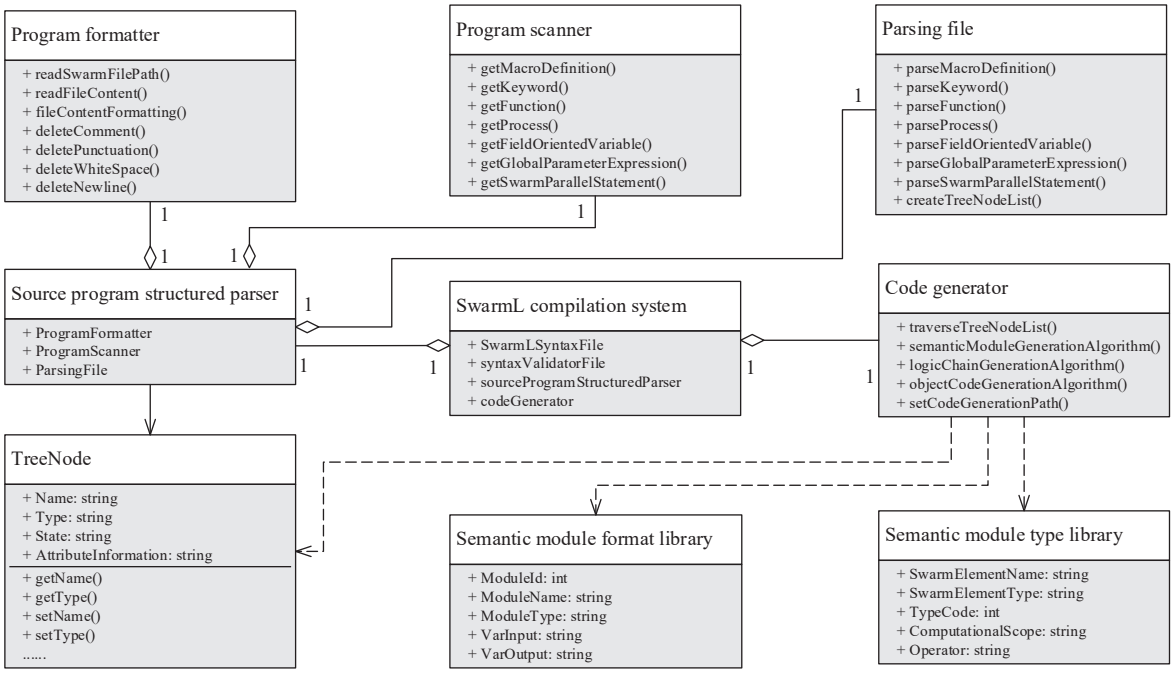


图 10 编译系统工具架构

本文将 SwarmL 编译系统和编辑环境集成, 实现了可视化的全分布式智能建筑系统应用程序一体化开发环境. 图 11 显示了利用该编译系统开发和编译 SwarmL 源程序的界面, 图左侧为变风量系统优化控制任务源程序. 点击菜单栏的编译按钮, 即自动生成了后缀名为*.json 的逻辑链中间代码文件. 图 11 左侧 SwarmL 程序段表示 1 个域定义语句和 1 个场变量定义语句, 图 11 右侧展示了编译系统自动生成的对应的标准化语义模块序列, 每条 SwarmL 语句对应 1 个语义模块, 每个语义模块定义了模块序号, 类型码, 名称, 输入输出信息. 编译系统工具的安装包以及实际操作演示视频已上传至本文补充材料[32]. 另外, 通过修改代码生成器文件可以为不同类型的群体智能系统硬件平台生成目标代码, 进而屏蔽硬件平台的异构性. 通过生成面向目标平台的代码来支持部署阶段, 从而产生一个由各个节点协作承载的群体智能系统.

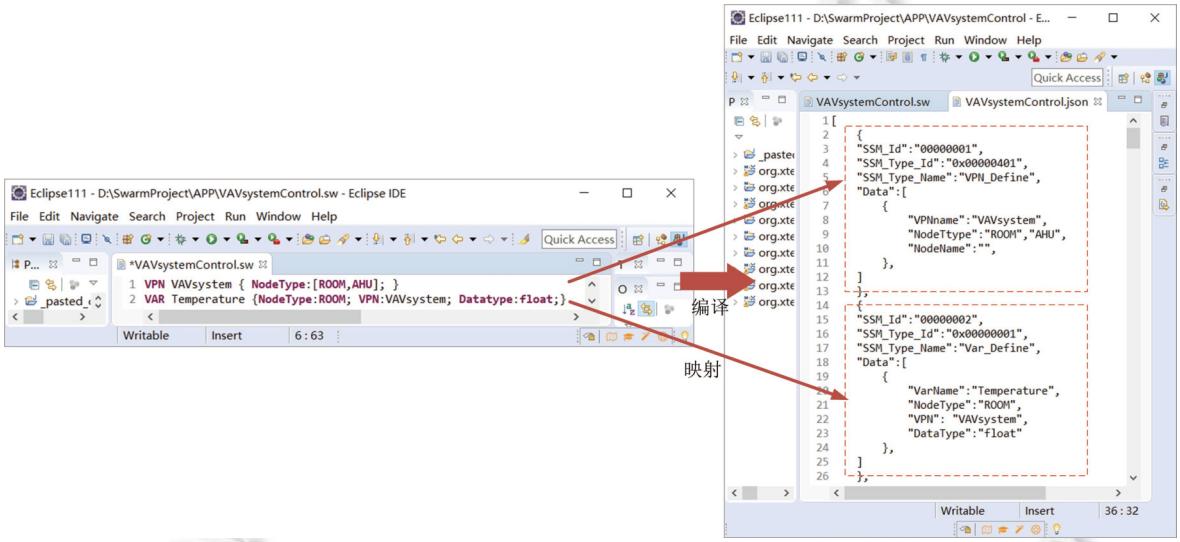


图 11 代码生成界面

4 实验评估

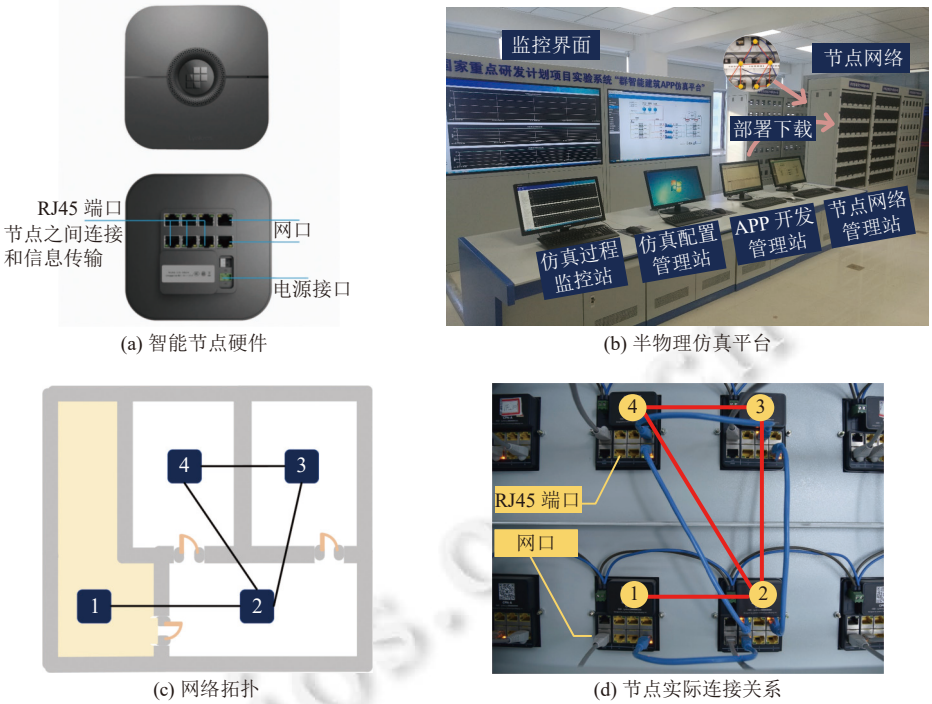
本节详细描述了实验评估过程中使用的目标平台、实验方法、比较指标和基准程序. 第 4.1 节搭建了实验仿真平台. 第 4.2 节通过测试编译系统所生成目标代码的执行效果验证了该编译系统的有效性. 第 4.3 节验证了本编译系统的应用程序开发效率以及目标代码运行性能.

4.1 实验平台搭建

实验平台建立在本研究团队所搭建的全分布式智能建筑系统应用程序半物理仿真平台[37], 其能够模拟真实的全分布式智能建筑系统, 进而支持应用程序的仿真、测试和验证. 该仿真平台使用 Lenovo Think Station P320 机器, 配备 8 核 Intel Core(TM) i7-7700 CPU, 主频为 3.60 GHz, 内存为 16 GB.

该平台包括一组智能节点, 所有节点的形状和尺寸都相同. 如图 12(a) 所示, 节点内置 TOS 专用操作系统[27], 并具有 1 个处理器、1 个存储器、2 个网络接口和 6 个 RJ45 接口. 节点可以通过 WiFi 与区域控制器 (DCU) 通信, 进而与本地控制器、传感器和执行器进行信息交互. RJ45 接口用于智能节点间的通信, 每个节点只与邻居节点通信, 通过与邻居的层层交互, 可以获取全局的信息, 从而构成一个扁平化、无中心的群体智能计算平台. 节点连接关系对应于建筑功能系统的虚拟拓扑关系. 图 12(b) 展示了仿真平台的现实架构图, 其中, 节点网络用于模拟建筑物理世界, 仿真配置管理站和节点网络管理站用于配置建筑仿真模型, APP 开发管理站用于开发建筑控制管理 APP 并将其部署到节点网络, 仿真过程监控站和监控界面用于监测 APP 的数据运行曲线.

© 中国科学院软件研究所 http://www.jos.org.cn



(a) 智能节点硬件 (b) 半物理仿真平台

(c) 网络拓扑 (d) 节点实际连接关系

图 12 全分布式智能建筑系统应用程序仿真平台环境搭建

4.2 编译系统有效性验证

基于全分布式智能建筑系统应用程序仿真平台, 本文对 SwarmL 编译系统生成的目标代码在该平台上的运行情况进行实验. 首先, 构建建筑功能系统网络拓扑, 进行空间划分和节点划分. 如图 12(c) 所示, 所仿真的建筑由 4 个区域组成, 数字表示智能节点的名称, 节点间的细实线表示网络的拓扑连接关系. 白色区域块代表居住者的主要活动区, 黄色区域块为空气处理机组所在区域. 图 12(d) 展示了节点在仿真平台中的实际连接关系. 然后, 建立火灾报警系统和变风量空调系统的设备仿真模型. 开发第 1.3 节中所描述的火灾报警信号全局扩散、房间需求风量全局求和以及邻域风阀校核应用程序, 并分别将其编译为目标代码, 下载到预先配置好的节点网络上执行进而仿真建筑控制管理应用程序的运行过程. 仿真平台与智能节点通过网络接口连接, 能够实时读取每个应用程序的运行结果.

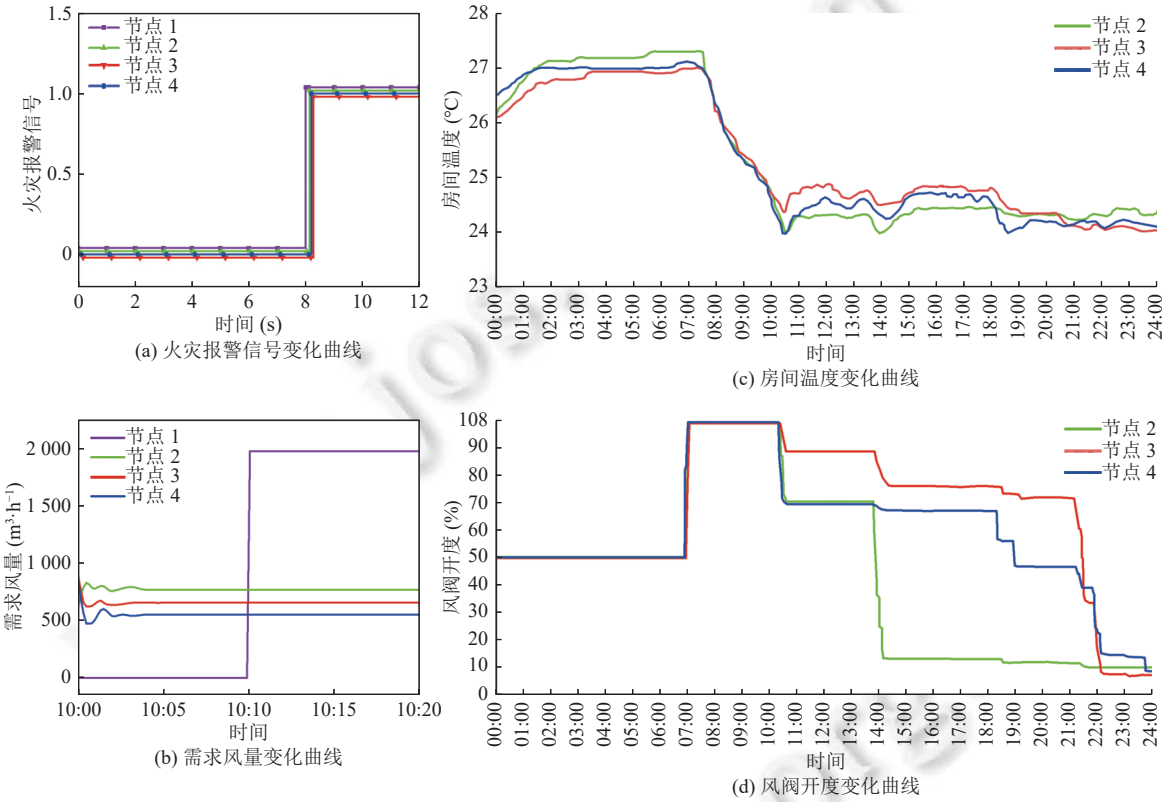
图 13(a) 展示了火灾报警信号全局扩散任务中各个节点上火灾报警信号值的变化曲线. 该任务采用条件触发机制, 当未发生火灾报警时各个节点的信号值为 0, 不满足应用程序触发条件, 火灾报警系统不动作. 8 s 时刻 1 号节点的火灾探测器探测到本区域发生火灾, 其信号值从 0 变为 1 发出火灾报警, 满足火灾报警信号全局扩散应用程序的触发条件. 此时 1 号节点作为发起点触发应用程序开始执行, 并向其余 3 个节点扩散报警信号. 随后所有节点的信号值都由 0 变为 1, 进而实现了火灾报警信号从发起点到系统中所有节点的扩散过程.

图 13(b) 给出了房间需求风量全局求和任务中各个节点上需求风量值的变化曲线, 其选取了 2023 年 7 月 15 日上午 10:00–10:20 的实验数据. 其中, 1 号节点是 AHU 节点, 而 2–4 号节点都是房间节点. 该任务采用周期性触发机制, 触发周期为 30 min. 当计算任务未触发时, 1 号节点的需求风量和的值 Vsum 为 0, 而 2–4 号节点的需求风量值为各自风阀采集到的需求风量. 在 10:00 时满足该任务的触发周期, 1 号节点作为发起点触发该应用程序执行, 并对所有房间节点的需求风量进行求和计算. 仿真结果显示 1 号节点的需求风量和的值由 0 变为 2–4 号节点的需求风量之和, 符合需求风量全局求和任务的运算逻辑.

图 13(c) 和图 13(d) 记录了 7 月 15 日 0:00–24:00, 各个节点的房间温度和风阀开度在邻域风阀校核应用程序的控制策略下的实际运行效果. 按照计划策略操作, 7:00 开空调, 12:00 关空调, 房间温度设定值为 25°C, 风阀开度

© 中国科学院软件研究所 http://www.jos.org.cn

初始值设置为 50%. 该任务采用条件触发机制, 其运行过程主要分为 5 个阶段. 阶段 1: 当温度超过设定温度+1°C 时, 7:00 时刻所有房间的风阀开度迅速达到 100%. 阶段 2: 10:25 时刻 2 号和 4 号房间的温度低于设定温度 1°C, 风阀开度降低. 3 号房间风阀开度也因邻居风阀降低的影响而变小. 阶段 3: 14:00 时刻 2 号房间温度比设定温度低 1°C, 风阀开度降低. 3 号和 4 号房间受邻居风阀的影响也降低风阀开度. 由于 3 号房间靠近 2 号房间风阀, 3 号房间风阀的调节范围较大. 阶段 4: 18:35 时刻 4 号房间温度再次低于设定温度 1°C, 风阀开度减小. 2 号和 3 号房间的风阀开度也因邻居风阀影响而变小. 阶段 5: 各房间温度变化不大, 风阀开度基本恒定. 可以看出在程序运行过程中, 每个房间都根据邻居节点传来的风阀变化量自动调节本地风阀开度, 调节过程平稳, 没有相互影响导致的振荡. 且房间温度始终控制在 24–25°C 之间, 保证了室内环境的舒适性.



(a) 火灾报警信号变化曲线　(c) 房间温度变化曲线

(b) 需求风量变化曲线　(d) 风阀开度变化曲线

图 13　目标代码运行结果

上述实验表明, 该编译系统能够将符合用户串行思维、声明式编程的 SwarmL 程序转换为节点独立执行的目标代码, 实现自组织、并行的建筑控制任务逻辑, 进而大大降低应用程序开发的工作量.

4.3　编译系统效率验证

本节通过与现有开发方法对比, 验证了所提出编译系统的应用程序开发效率以及目标代码运行性能.

4.3.1　应用程序开发效率对比

编译群体智能应用程序过程的复杂性导致应用程序开发过程困难, 开发效率不高. 本文提出群体智能应用程序编译系统的主要目的是解决此类程序在开发过程中编译环节的技术挑战, 打通群体智能应用程序开发的关键堵点, 进而缩短应用程序从编程到编译全过程开发时间, 提升开发效率. 由于当前计算机运行计算能力足够强, 各类编译系统编译源程序的时间都能够维持在较短时间内, 并没有显著差距, 单纯比较编译时间的微小差别似乎没有决定性意义. 因此, 本实验重点评估比较编译系统支撑下的应用程序开发效率, 并没有直接评估编译基准程序的效率.

© 中国科学院软件研究所 http://www.jos.org.cn

根据目标问题度量 (GQM) 方法[38]评估 SwarmL 在应用程序开发中的效率, 采用应用程序开发时间作为评估指标, 将 SwarmL 编译系统与应用于全分布式智能建筑系统的 TOS 图形化编程系统[39]和 Touch 语言编程系统[40]在应用程序开发效率方面进行对比. TOS 图形化编程系统和 Touch 语言编程系统是前期主要用于全分布式智能建筑系统应用程序开发的编程工具. 其中 TOS 图形化编程系统将变量定义、函数调用等元素封装成图元块, 开发者可以通过拖拽图元块实现应用程序开发. Touch 语言是一种文本化编程语言, 其采用“基本单元, 邻域, 域”的领域特定编程模型实现对全分布式智能建筑系统应用程序的敏捷开发. 本节讨论了实验规划与设计, 包括实验对象、实验设计、实验变量选择和实验过程.

(1) 实验对象

实验对象是参加软件工程研究生课程的 15 名研究生. 所有参与者都学习过面向对象编程、软件工程和分布式系统, 并至少有使用 C、Java、Visual Studio 和 Eclipse IDE 开发程序的基本经验以及全分布式智能建筑系统的基本知识.

(2) 实验设计

本实验选取的 15 名实验对象的学术水平和编程能力基本一致. 按照随机化和平衡的原则, 实验对象被分为 3 组, 每组 5 人. 第 1 组使用 TOS 图形化编程系统, 第 2 组使用 Touch 语言编程系统, 第 3 组使用 SwarmL 语言及其编译系统. 考虑到变风量系统优化控制任务基本涵盖所有分布式建筑系统控制逻辑 (例如本地计算, 邻域交互, 全局求和等), 3 组实验对象均被要求开发变风量系统优化控制应用程序.

(3) 实验变量

3 组实验变量定义如下.

控制变量: 3 组实验对象开发相同的变风量系统优化控制应用程序.

自变量: 应用程序开发方法.

因变量: 应用程序开发时间.

(4) 实验过程

在实验任务开始之前, 对实验对象进行一次 3 h 的会议培训, 内容包括: (a) 详细介绍 SwarmL 语言及其编译系统, TOS 图形化编程系统, Touch 语言编程系统的使用方法和程序案例, 并支持实验对象使用不同开发方法练习编程和编译全分布式智能建筑系统应用程序. (b) 解释变风量系统分布式优化控制任务的算法逻辑. 实验对象被告知要在确保正确的情况下尽量快速完成对变风量系统分布式优化控制应用程序的开发. 完成开发任务后, 所有实验对象使用教学平台提交代码, 进而记录每人开发应用程序所用的时间.

(5) 结果分析

使用 t 检验确定开发方法 (自变量) 对开发时间 (因变量) 的影响. 在 α=0.05 时进行 t 检验. 设置假设如下.

零假设 (H0): 对于同一任务, SwarmL 语言及其编译系统与 Touch 编程系统不存在显著差异.

备择假设 (H1): 对于同一任务, SwarmL 语言及其编译系统与 Touch 编程系统存在显著差异.

零假设 (H2): 对于同一任务, SwarmL 语言及其编译系统与 TOS 编程系统不存在显著差异.

备择假设 (H3): 对于同一任务, SwarmL 语言及其编译系统与 TOS 编程系统存在显著差异.

表 2 给出应用程序开发时间的平均值和标准差. 图 14 显示了开发时间的箱型图. 表 3 给出零假设的 t 检验结果. 每个测试产生的结果都小于统计显著性水平 5%. 因此, 本实验的结论是, SwarmL 语言及其编译系统与 Touch 编程系统和 TOS 编程系统在开发时间上的差异是显著的.

表 2 应用程序开发时间对比 (min)

开发方法	平均开发时间	标准差
Touch编程系统	123.00	11.51
TOS编程系统	92.00	6.23
SwarmL语言及其编译系统	76.40	5.16

© 中国科学院软件研究所 http://www.jos.org.cn

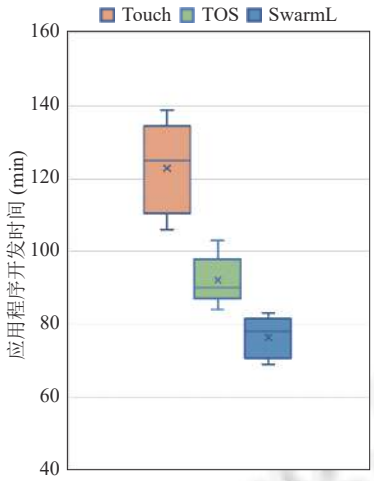


图 14 应用程序开发时间的箱型图

表 3 应用程序开发时间对零假设的 t 检验结果

标准	t	df	p	结论
SwarmL组和Touch组在开发时间方面的零假设	5.72	4	2.3×10^{-3}	在5%显著性水平上拒绝原假设 ($p<0.05$)
SwarmL组和TOS组在开发时间方面的零假设	4.78	4	4.3×10^{-3}	

(6) 讨论

图 14 展示了使用不同方法开发全分布式智能建筑系统应用程序的平均开发时间. 结果表明, 使用 Touch 编程系统开发应用程序的平均时间最长, 达到 123 min. 分析实验过程发现, 主要原因在于 Touch 语言仅支持从邻域角度定义变量和描述分布式计算逻辑, 开发者需要花费时间考虑邻居间的复杂交互行为来编程全局计算任务, 导致编程过程变得复杂. 另外, 其采用位置量的方式刻画变量的分布属性, 描述邻域计算时需要一一指定每个邻居节点的编号, 进而降低了开发效率. 与 Touch 编程系统相比, 使用 TOS 图形化编程系统的平均开发时间有所降低, 为 92 min. TOS 图形化编程系统允许开发者使用拖拽图元块的方式开发应用程序, 具有较好的简洁性. 然而, 其并不能完全支持图形化编程, 开发者仍需要在图形块的文本编辑界面手动定义一些分布式计算函数或者函数调用信息, 导致编程工作量大大增加. 使用 SwarmL 及其编译系统的平均开发时间最短, 为 76.4 min. 一个重要的原因在于, SwarmL 所定义的场变量将分布式建筑物理场和建筑控制任务的并行计算模式深度融合, 允许用户以串行思维方式从全局角度编程分布式计算任务, 而无需考虑底层通讯协议和数据分配等复杂细节. 因此, 实验结果表明, 与现有开发方法相比, SwarmL 及其编译系统能够提升全分布式智能建筑系统应用程序的开发效率, 有效降低开发时间.

4.3.2 目标代码运行性能对比

评估编译系统性能的一个重要指标是目标代码的执行效率. 本节将 SwarmL 编译系统与应用于全分布式智能建筑系统的 TOS 图形化编译器[39]和 Touch 语言编译器[40]在目标代码的性能方面进行对比. 选取 10 个典型的全分布式智能建筑系统控制管理任务作为基准应用程序, 其已经被实际开发并应用在国家重点研发计划项目示范工程[27,41]中, 包括变风量自组织优化控制 vavControl、人员疏散 fireEvacuation、湿度传感器故障诊断 faultDiagnosis、仓库管理 storeManagement、基于人员分布的照明控制 lightControl、冷冻水管网末端检测 checkEnd、能耗统计 energyConsumption、冷机群控 chillerControl、管网流量识别 flowIdentification、并联水泵自组织优化控制 pumpControl. 这组基准程序都基于邻居节点并行交互实现整体层面的建筑控制管理任务, 且涵盖常用的群体智能计算特征, 例如全局扩散, 全局求和, 邻域校核等. 表 4 给出了 10 个基准程序的功能描述.

© 中国科学院软件研究所 http://www.jos.org.cn

分别使用 TOS 图形化编程系统, Touch 语言和 SwarmL 语言编写上述基准程序. 然后使用 TOS 图形化编译器、Touch 语言编译器和 SwarmL 编译系统编译这 3 个版本的基准程序并生成目标代码, 将目标代码部署到全分布式智能建筑系统仿真平台上运行. 对于每个基准程序, 重复执行 10 次, 记录平均运行时间, 并比较它们在 Touch 语言版本应用程序运行时间 (归一化为 1) 上的加速比. 图 15 的实验结果表明, 相比于 Touch 语言编译器, SwarmL 语言编译系统生成的目标代码平均加速比可达 1.527, 实现了性能提升. 本文分析这主要是由于 SwarmL 编译系统将程序中的复杂表达式编译成嵌套的多个 if 语句代码, 并对部分循环进行合并, 尽量避免不必要的循环分支和计算, 进而减少了循环次数和性能开销. 另外, 与 TOS 图形化编译器相比, SwarmL 语言编译系统也具备更好的性能. TOS 图形化编译器引入了较多的函数调用, 如参数传递, 这导致一些额外性能开销增加. 而 SwarmL 编译系统将一些计算逻辑内联到循环中, 减少函数调用次数, 一定程度上提升了目标代码运行性能.

表 4 基准程序的功能描述

基准程序	描述
vavControl	多节点协作计算求取房间风阀开度和AHU风机转速
fireEvacuation	通过并行递归法计算疏散时间最短的逃生方向和逃生路径
faultDiagnosis	通过邻居节点交互查找发生故障的传感器节点
storeManagement	实现货物查询、存储和自动导向车 (AGV)路径规划
lightControl	根据人员数量和房间照度的变化控制照明设备状态
checkEnd	根据节点协作调节水泵转速, 使得末端管网阻力最小, 降低水泵能耗
energyConsumption	根据节点协作计算整个建筑系统中的能耗
chillerControl	多台并联冷机的分配优化任务
flowIdentification	根据流体网络中安装的压力传感器, 对流体网络各管段的流量进行辨识计算
pumpControl	多台水泵协作计算效率最高时的开启台数和频率, 使得系统总体能效最优

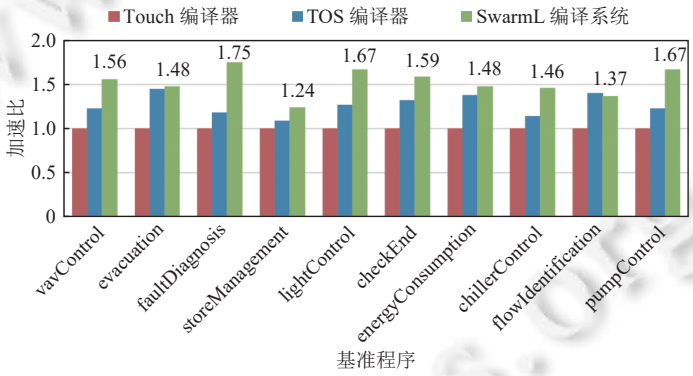


图 15 基准程序运行性能对比

以上工作是与面向全分布式智能建筑的同类编译系统的性能对比. 在采用分布存储结构的其他相关工作中, 面向分布式 MPI 的并行化编译系统[21,22]与本文内容较为类似. 基于 MPI 的分布式系统采用广播和收集模式进行通信, 并行化编译产生的通信代码需要遍历系统中所有节点个体或者进程并发送广播消息. 而群体智能系统中个体的通信范围仅限于邻居节点, 通过相邻节点局部交互实现全局层面的群体智能计算任务. 由于全分布式智能建筑系统中每个节点实际连接的邻居数量最多不超过 6 个, 从通信机制和循环遍历规模的角度定性分析, 本文编译系统产生的通信代码规模和开销比面向 MPI 的并行化编译系统更小, 程序执行效率也将更高.

5 相关工作

由于群体智能系统和多核处理器系统对并行程序的迫切需求以及编程并行程序的复杂性, 并行化编译技术成

© 中国科学院软件研究所 http://www.jos.org.cn

为当前并行计算领域的一个研究热点[9].

相关研究提出大量源代码到源代码的并行化编译系统 (例如编译器、工具和库) 以实现代码的自动并行化[13,14]. 并行化编译系统能够自动检测串行程序中潜在的并行性并将串行程序转变为等价的适合并行体系结构运行的并行程序[42], 其涉及到数据依赖关系分析、程序变换、数据分布等技术. SUIF 编译器[43]是首个用于 C 和 Fortran 语言的顺序到并行代码自动转换工具, 能够自动将用 C 或 Fortran 编写的顺序密集矩阵计算转换为具有共享内存的机器的并行代码. 用于 ANSI C 程序的源对源编译器 Cetus[44], 通过私有和共享变量分析以及 OpenMP 指令的自动插入提供循环的自动并行化. Pluto[45]是一个基于多面体模型[46]的自动并行化工具, 对 C 程序执行源到源转换, 以同时实现粗粒度或细粒度的并行性和数据局域性. 基于 OpenMP 的自动并行化编译器 AutoPar-Clava[42], 支持静态检测 C 应用程序中的并行化循环, 并根据访问模式对目标循环内使用的变量进行分类, 从输入的顺序程序版本生成 C OpenMP 并行代码. 上述方法大多面向共享内存多核系统, 通过对程序循环实施合理的等价变换来消除阻碍循环并行化的依赖关系, 提高循环可并行执行的粒度. 本文所提出的编译系统面向分布存储结构, 除了要解决并行粒度的问题, 还需处理群体智能并行任务识别和分配调度等问题.

面向分布存储结构的并行化编译技术需要生成消息传递通信代码以支持访问分布在其他进程 (或个体) 上的数据, 进而支持个体间的交互机制. MPI 自动并行化编译系统[47]是一种面向分布式存储系统的源源变换软件, 能够将串行程序正确地编译为使用消息传递接口的单程序多数据 (SPMD) 并行程序. 文献 [22] 基于 Open64 的 MPI 自动并行化编译系统后端, 提出一种消息传递代码生成算法. 从统一数据分布的角度, 根据给定的并行化循环集和通信数组集, 通过修改 WHIRL (Open64 编译器中间表示语言) 表示的串行代码语法结构树, 使串行代码自动生成更精确的消息传递代码, 进而提升消息传递程序的加速比. HELIX-RC 编译框架[48]通过将循环迭代编译为并行线程, 将同步和数据通信从计算中解耦, 能够有效解决顺序程序中的数据依赖限制并行化的问题, 使得非数值程序更容易自动并行化, 并降低通信开销. 与这些工作不同, 本文编译系统产生的通信代码仅限于相邻节点间交互通信而不是全局广播, 因而在计算中需要循环遍历节点的次数减少, 生成的通信代码规模和开销更小, 具有更好的运行性能.

另外, 设计合适、通用化的编译系统中间表示能够使得并行化编译系统不再局限于特定目标语言、特性和平台, 并提升编译系统转换和优化的效率[49]. PENCIL[50]是一种平台中立、可移植的计算中间语言, 用于直接加速器编程和领域特定语言编译. PENCIL 使许多形式的非静态仿射代码和访问模式能够适应多面体框架内的高级循环转换和并行化. SPIRE[51]是一种新的并行语言 IR 设计方法, 其支持以系统的方式将编译平台中使用的任何顺序 IR 映射为并行 IR, 已用于自动 OpenMP 和 MPI 任务级并行化. 面向异构硬件的机器学习编译器 Glow[52]将传统的神经网络数据流图降低为两阶段强类型的中间表示 IR. 高级 IR 允许优化器执行特定于领域的优化, 低级的基于指令的纯地址 IR 允许编译器执行与内存相关的优化. 这使得适合每个 IR 级别的分析和优化能够有效地、可扩展地针对多个后端. 与上述工作相比, 本文所提出的编译系统中间层通过标准化语义模块实现了群体智能计算类型和数据的解耦, 具有较好的普适性, 可应用至不同类型的群体智能硬件平台.

6 总 结

本文提出一种用于编译群体智能系统应用程序的编译系统. 针对符合用户串行思维、面向系统全局编程的群体智能应用程序和并行的节点可执行目标代码之间的语义鸿沟, 建立了编译系统总体框架. 通过源程序并行信息识别, 计算划分, 交互信息生成技术, 提出了面向系统全局、串行、声明式编程的群体智能系统应用程序到节点可独立执行的并行目标代码的编译算法. 针对目标硬件平台的异构性, 提出了一种具有标准化格式的中间代码, 将复杂群体智能计算任务转换为由群体智能算子和输入输出变量组成的标准化语义模块, 其以独立于平台的形式表示源程序信息, 支持向不同结构硬件平台目标代码的映射. 实现了具有应用程序编辑和编译功能的用户友好的编译系统工具. 实验测试分别从编译系统有效性, 应用程序开发效率和性能测试 3 个方面进行了验证, 结果表明, 该编译系统能够将串行的群体智能系统应用程序转换为硬件平台可执行的并行目标代码, 且提升应用程序开发效率,

© 中国科学院软件研究所 http://www.jos.org.cn

其生成的代码在一系列基准测试中具有比现有编译器更好的性能.

基于该编译系统, 用户能够以串行方式来全局声明式地编程群体智能应用程序, 隐式地规范同步和通信, 因此节点间的数据移动、并行任务管理等复杂细节完全留给编译系统, 这有效降低了群体智能应用程序编程和调试的难度, 提高了群体智能系统的开发效率. 此外, 本文这种将群体智能应用程序编译为标准化中间表示进而屏蔽硬件平台异构性的思想方法, 也具备一定的普适性, 对其他领域特定语言的编译开发具有借鉴意义. 在未来, 将在并行循环优化、常量折叠等方面探索该编译系统的代码优化方法, 提高该编译系统生成的目标代码的执行效率.

References:

[1] Hassanien AE, Emary E. Swarm Intelligence: Principles, Advances, and Applications. Boca Raton: CRC Press, 2018. [doi: 10.1201/9781315222455]

[2] Xing T, Yan H, Sun KL, Wang YF, Wang XT, Zhao QC. Honeycomb: An open-source distributed system for smart buildings. Patterns, 2022, 3(11): 100605. [doi: 10.1016/j.patter.2022.100605]

[3] Zedadra O, Guerrieri A, Jouandeau N, Spezzano G, Seridi H, Fortino G. Swarm intelligence-based algorithms within IoT-based systems: A review. Journal of Parallel and Distributed Computing, 2018, 122: 173–187. [doi: 10.1016/j.jpdc.2018.08.007]

[4] Youssefi KAR, Rouhani M, Mashhadi HR, Elmenreich W. A swarm intelligence-based robotic search algorithm integrated with game theory. Applied Soft Computing, 2022, 122: 108873. [doi: 10.1016/j.asoc.2022.108873]

[5] Schranz M, Di Caro GA, Schmickl T, Elmenreich W, Arvin F, Şekercioğlu A, Sende M. Swarm intelligence and cyber-physical systems: Concepts, challenges and future trends. Swarm and Evolutionary Computation, 2021, 60: 100762. [doi: 10.1016/j.swevo.2020.100762]

[6] Chen WJ, Yang QL, Jiang ZY, Xing JC, Zhao S, Zhou QZ, Han DS, Feng BW. SwarmL: A language for programming fully distributed intelligent building systems. Buildings, 2023, 13(2): 499. [doi: 10.3390/buildings13020499]

[7] Pinciroli C, Beltrame G. Buzz: A programming language for robot swarms. IEEE Software, 2016, 33(4): 97–100. [doi: 10.1109/MS.2016.95]

[8] Modi R. Solidity Programming Essentials: A Beginner's Guide to Build Smart Contracts for Ethereum and Blockchain. Birmingham: Packt Publishing, 2018.

[9] Midkiff SP. Automatic Parallelization: An Overview of Fundamental Compiler Techniques. Cham: Springer, 2012. [doi: 10.1007/978-3-031-01736-0]

[10] Arabnejad H, Bispo J, Cardoso JMP, Barbosa JG. Source-to-source compilation targeting OpenMP-based automatic parallelization of C applications. The Journal of Supercomputing, 2020, 76(9): 6753–6785. [doi: 10.1007/s11227-019-03109-9]

[11] Jiang S, Cao JN, Liu Y, Chen JL, Liu XF. Programming large-scale multi-robot system with timing constraints. In: Proc. of the 25th Int'l Conf. on Computer Communication and Networks. Waikoloa: IEEE, 2016. 1–9. [doi: 10.1109/ICCCN.2016.7568563]

[12] Ashley-Rollman MP, Lee P, Goldstein SC, Pillai P, Campbell JD. A language for large ensembles of independently executing nodes. In: Proc. of the 25th Int'l Conf. on Logic Programming. Pasadena: Springer, 2009. 265–280. [doi: 10.1007/978-3-642-02846-5_24]

[13] Prema S, Nasre R, Jehadeesan R, Panigrahi BK. A study on popular auto-parallelization frameworks. Concurrency and Computation: Practice and Experience, 2019, 31(17): e5168. [doi: 10.1002/cpe.5168]

[14] Ma CY, Lv BX, Ye XJ, Zhang Y. Automatic parallelization framework for complex nested loops based on LLVM pass. Ruan Jian Xue Bao/Journal of Software, 2023, 34(7): 3022–3042 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6858.htm [doi: 10.13328/j.cnki.jos.006858]

[15] Intel C++ Compiler. 2024. https://software.intel.com/en-us/c-compilers/

[16] Legacy PGI Support. 2024. http://www.pgroup.com

[17] Bhosale A, Barakhshan P, Rosas MR, Eigenmann R. Automatic and interactive program parallelization using the Cetus source to source compiler infrastructure v2.0. Electronics, 2022, 11(5): 809. [doi: 10.3390/electronics11050809]

[18] Lidman J, Quinlan DJ, Liao C, McKee SA. Rose: Fttransform-a source-to-source translation framework for exascale fault-tolerance research. In: Proc. of the 2012 IEEE/IFIP Int'l Conf. on Dependable Systems and Networks Workshops. IEEE, 2012. 1–6. [doi: 10.1109/DSNW.2012.6264672]

[19] Palkowski M, Bielecki W. TRACO: Source-to-source parallelizing compiler. Computing and Informatics, 2016, 35(6): 1277–1306.

[20] Bondhugula U, Hartono A, Ramanujam J, Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. In: Proc. of the 29th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Tucson: ACM, 2008. 101–113. [doi: 10.1145/

© 中国科学院软件研究所 http://www.jos.org.cn

1375581.1375595]

[21] Ferner CS. The Paraguin compiler-message-passing code generation using SUIF [Stanford University Intermediate Format]. In: Proc. of the 2002 IEEE SoutheastCon. Columbia: IEEE, 2002. 1–6. [doi: 10.1109/SECON.2002.995545]

[22] Chen DZ, Zhao RC, Yao Y, Han L. Message-passing code generation algorithm in the MPI automatic parallelizing compilation system. Computer Science, 2012, 39(6): 301–304 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2012.06.075]

[23] Jordan H, Pellegrini S, Thoman P, Kofler K, Fahringer T. INSPIRE: The insieme parallel intermediate representation. In: Proc. of the 22nd Int'l Conf. on Parallel Architectures and Compilation Techniques. Edinburgh: IEEE, 2013. 7–17. [doi: 10.1109/PACT.2013.6618799]

[24] Schardl TB, Moses WS, Leiserson CE. Tapir: Embedding fork-join parallelism into LLVM's intermediate representation. In: Proc. of the 22nd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Austin: ACM, 2017. 249–265. [doi: 10.1145/3018743.3018758]

[25] Armstrong TG, Wozniak JM, Wilde M, Foster IT. Compiler techniques for massively scalable implicit task parallelism. In: Proc. of the 2014 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. New Orleans: IEEE, 2014. 299–310. [doi: 10.1109/SC.2014.30]

[26] Shen Q. Studies on architecture of decentralized system in intelligent building [Ph.D. Thesis]. Beijing: Tsinghua University, 2015 (in Chinese with English abstract).

[27] Zhao QC, Xai L, Jiang ZY. Project report: New generation intelligent building platform techniques. Energy Informatics, 2018, 1: 1–5. [doi: 10.1007/s42162-018-0011-9]

[28] Zhao QC, Jiang ZY. Insect intelligent building (I^2B): A new architecture of building control systems based on Internet of Things (IoT). In: Proc. of the 2019 Int'l Conf. on Smart City and Intelligent Building. Singapore: Springer, 2019. 457–466. [doi: 10.1007/978-981-13-6733-5_42]

[29] Jiang ZY, Dai YC. A decentralized, flat-structured building automation system. Energy Procedia, 2017, 122: 68–73. [doi: 10.1016/j.egypro.2017.07.285]

[30] Dai YC, Jiang ZY, Shen Q, Chen PZ, Wang SQ, Jiang Y. A decentralized algorithm for optimal distribution in HVAC systems. Building and Environment, 2016, 95: 21–31. [doi: 10.1016/j.buildenv.2015.09.007C]

[31] Yu JQ, Liu QT, Zhao AJ, Qian XG, Zhang R. Optimal chiller loading in HVAC system using a novel algorithm based on the distributed framework. Journal of Building Engineering, 2020, 28: 101044. [doi: 10.1016/j.jobe.2019.101044]

[32] SwarmIntelligenceCompiler. 2024. https://gitee.com/resiliencewj/supplemental-material

[33] Bernstein PA. Applying model management to classical meta data problems. In: Proc. of the 1st Biennial Conf. on Innovative Data Systems Research. Asilomar: CIDR, 2003. 209–220.

[34] Nastic S, Sehic S, Vögler M, Truong H L, Dustdar S. PatRICIA—A novel programming model for IoT applications on cloud platforms. In: Proc. of the 6th Int'l Conf. on Service-Oriented Computing and Applications. Koloa: IEEE, 2013. 53–60. [doi: 10.1109/SOCA.2013.48]

[35] Dedousis D, Kalogeraki V. A framework for programming a Swarm of UAVs. In: Proc. of the 11th Pervasive Technologies Related to Assistive Environments Conf. Corfu: ACM, 2018. 5–12. [doi: 10.1145/3197768.3197772]

[36] Bettini L. Implementing Domain-specific Languages with Xtext and Xtend. 2nd ed., Birmingham: Packt Publishing, 2016.

[37] Liu XG, Xie LQ, Yang QL, Xing JC, Jiang ZY, Zhao QC. Simulation experiment study on firefighting system under the insect intelligent building platform. Building Science, 2020, 36(4): 149–154 (in Chinese with English abstract). [doi: 10.13614/j.cnki.11-1962/tu.2020.04.22]

[38] Koziolek H. Goal, question, metric. Dependability Metrics: Advanced Lectures, 2008, 39–42. [doi: 10.1007/978-3-540-68947-8_6]

[39] Zhu DD. Building automation platform based on insect intelligence. Building Energy Efficiency, 2018, 46(11): 1–4 (in Chinese with English abstract). [doi: 10.3969/j.issn.1673-7237.2018.11.001]

[40] Chen WJ, Yang QL, Jiang ZY, Xing JC, Zhao QC, Zhou QZ, Han DS. Touch: A textual programming language for developing APPs of insect intelligent building. Scientific Programming, 2020, 2020: 8887588. [doi: 10.1155/2020/8887588]

[41] Dai YC, Jiang ZY, Zhang F, Li HB, Zhao XH, Song ML, Pu J. Application of Swarm intelligent system to a new commercial building. Heating Ventilating & Air Conditioning, 2019, 49(11): 18–25, 17 (in Chinese with English abstract).

[42] Arabnejad H, Bispo J, Barbosa JG, Cardoso JMP. AutoPar-clava: An automatic parallelization source-to-source tool for C code applications. In: Proc. of the 9th Workshop and the 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms. Manchester: ACM, 2018. 13–19. [doi: 10.1145/3183767.3183770]

© 中国科学院软件研究所 http://www.jos.org.cn

[43] Hall MW, Anderson JM, Amarasinghe SP, Murphy BR, Liao SW, Bugnion E, Lam MS. Maximizing multiprocessor performance with the SUIF compiler. Computer, 1996, 29(12): 84–89. [doi: 10.1109/2.546613]

[44] Bae H, Mustafa D, Lee JW, Aurangzeb, Lin H, Dave C, Eigenmann R, Midkiff SP. The cetus source-to-source compiler infrastructure: Overview and evaluation. Int'l Journal of Parallel Programming, 2013, 41(6): 753–767. [doi: 10.1007/s10766-012-0211-z]

[45] Bondhugula U, Baskaran M, Krishnamoorthy S, Ramanujam J, Rountev A, Sadayappan P. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In: Proc. of the 17th Int'l Conf. on Compiler Construction. Budapest: Springer, 2008. 132–146. [doi: 10.1007/978-3-540-78791-4_9]

[46] Khan AA, Mewes H, Grosser T, Hoefler T, Castrillon J. Polyhedral compilation for racetrack memories. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2020, 39(11): 3968–3980. [doi: 10.1109/TCAD.2020.3012266]

[47] Rauber T, Rünger G. Parallel Programming: For Multicore and Cluster Systems. Berlin: Springer, 2010. [doi: 10.1007/978-3-642-04818-0]

[48] Campanoni S, Brownell K, Kanev S, Jones TM, Wei GY, Brooks D. HELIX-RC: An architecture-compiler co-design for automatic parallelization of irregular programs. In: Proc. of the 41st Int'l Symp. on Computer Architecture (ISCA). Minneapolis: IEEE, 2014. 217–228. [doi: 10.1109/ISCA.2014.6853215]

[49] Susungi A, Tadonki C. Intermediate representations for explicitly parallel programs. ACM Computing Surveys, 2022, 54(5): 100. [doi: 10.1145/3452299]

[50] Baghdadi R, Beaugnon U, Cohen A, Grosser T, Kruse M, Reddy C, Verdoolaege S, Betts A, Donaldson AF, Ketema J, Absar J, van Haastregt S, Kravets A, Lokhmotov A, David R, Hajiyev E. PENCIL: A platform-neutral compute intermediate language for accelerator programming. In: Proc. of the 2015 Int'l Conf. on Parallel Architecture and Compilation. San Francisco: IEEE, 2015. 138–149. [doi: 10.1109/PACT.2015.17]

[51] Khaldi D, Jouvelot P, Irigoin F, Ancourt C. SPIRE: A methodology for sequential to parallel intermediate representation extension. In: Proc. of the 17th Workshop on Compilers for Parallel Computing. Lyon: CPC, 2013.

[52] Rotem N, Fix J, Abdulrasool S, Catron G, Deng S, Dzhabarov R, Gibson N, Hegeman J, Lele M, Levenstein R, Montgomery J, Maher B, Nadathur S, Olesen J, Park J, Rakhov A, Smelyanskiy M, Wang M. Glow: Graph lowering compiler techniques for neural networks. arXiv:1805.00907, 2018. [doi: 10.48550/arXiv.1805.00907]

附中文参考文献:

[14] 马春燕, 吕炳旭, 叶许姣, 张雨. 基于 LLVM Pass 的复杂嵌套循环自动并行化框架. 软件学报, 2023, 34(7): 3022–3042. http://www.jos.org.cn/1000-9825/6858.htm [doi: 10.13328/j.cnki.jos.006858]

[22] 陈达智, 赵荣彩, 姚远, 韩林. MPI 自动并行化编译系统中消息传递代码生成算法. 计算机科学, 2012, 39(6): 301–304. [doi: 10.3969/j.issn.1002-137X.2012.06.075]

[26] 沈启. 智能建筑无中心平台架构研究 [博士学位论文]. 北京: 清华大学, 2015.

[37] 刘夕广, 谢立强, 杨启亮, 邢建春, 姜子炎, 赵千川. 消防系统在群智能建筑平台下的仿真实验研究. 建筑科学, 2020, 36(4): 149–154. [doi: 10.13614/j.cnki.11-1962/tu.2020.04.22]

[39] 朱丹丹. 群智能建筑控制平台技术. 建筑节能, 2018, 46(11): 1–4. [doi: 10.3969/j.issn.1673-7237.2018.11.001]

[41] 代允闯, 姜子炎, 张烽, 李洪波, 赵小虎, 宋媚琳, 浦江. 群智能系统在某商业建筑中的应用. 暖通空调, 2019, 49(11): 18–25, 17.

附录 A

图 A1 给出 SwarmL 抽象语法的核心部分. 其中, DomainDef 定义了域的名称, 并指定组成该域的约束条件, 包括节点类型、节点名称等. VarFieldDef 定义对应于建筑物理场的场变量名称, 并指定该变量对应的节点类型和域名, 其描述了场变量有效数值的分布范围. 参数表示场变量投影到某个或某些节点上的数值. ParaDef 定义参数名称, 并且指定该参数所指向的场变量名称, 以及参数的更新时间和计算域. FunctionDef 定义函数名称, 输出输入参数的名称和计算域. PeriodicTrigger 描述该某个建筑应用任务的起始时刻、结束时刻和触发周期, 其中触发周期为此进程内子语句执行的间隔周期. SwarmParallelStmt 通过指定节点类型和域名来约束哪些节点参与该语句的执行. 当建筑功能系统中某个节点作为发起点触发计算后, 所有满足该约束条件的节点都并行处理该语句. SwarmL 语法混合了命令式和函数式构造, 其中一部分类似于通用编程语言, 例如循环、分支和部分表达式等, 这确保了新手的学习曲线很短, 并且语言易于阅读和熟悉.

© 中国科学院软件研究所 http://www.jos.org.cn

program == *DomainDef VarFieldDef ParaDef FunctionDef**

DomainDef == **VPN** *VpnName* { **NodeType***; *ConditionExpr**; *NodeName** }

VarFieldDef == **VAR** *VarName* { **NodeType***; *VpnName**; **DataType** }

ParaDef == **DataType** *ParaName* (**Refer to:** *VarName*; *UpdateTime*; **@** *Scope*)

FunctionDef == [(**DataType** *OutputPara* (*@Scope*))*] = *FuncName* (**DataType** *InputPara* (*@Scope*))*) {*FuncStmt*}

FuncStmt == (*process* | *VarFieldDef* | *ParaDef* | *stmt*)*

process == *PeriodicTrigger* | *ConditionTrigger*

PeriodicTrigger == **Timer** (*StartTime* ; *EndTime* ; *Interval*) {*stmt*}

ConditionTrigger == **Condition** (*StartTime* ; *EndTime* ; *ConditionExpr*) {*stmt*}

stmt == *SwarmParallelStmt* | *ifStmt* | *forStmt* | *whileStmt* | *FuncCallExpr* | *expr*

SwarmParallelStmt == **Swarm** (**NodeType***; *VpnName*;) {*SubStmt*}

expr == *expr Operator expr* | *Operator exp* | *FuncCallExpr* | *ConditionExpr* | *exp*

FuncCallExpr == *FuncName* (*InputParaName**)

Scope == **StartNode** | **All** | **Me** | **AllNeighbors** | **UpStream** | **DownStreams**

NodeType == **ROOM** | **CHILLER** | **PUMP** | **BOILER** | **C_TOWER** | **AHU** | **CABINET** | …

图 A1　SwarmL 语言抽象语法

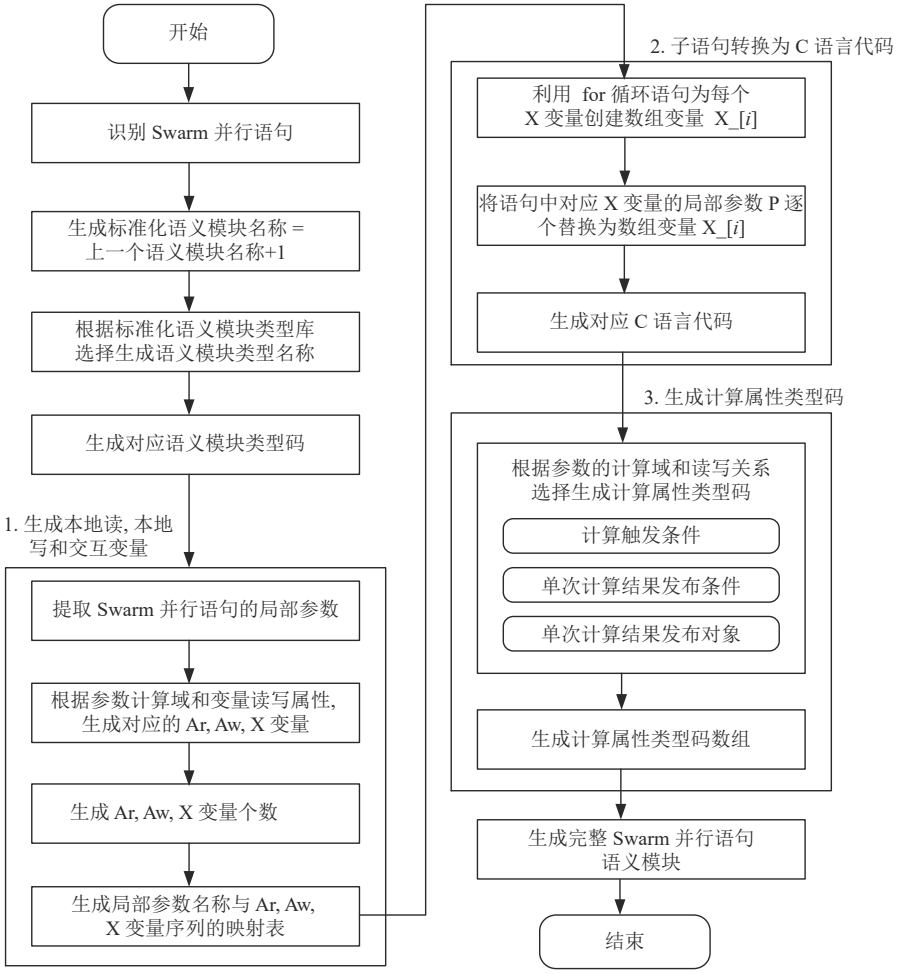


图 A2　Swarm 并行语句编译算法流程图

陈文杰(1996－), 男, 博士生, 主要研究领域为软件工程, 智能建筑.

杨启亮(1975－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 智能建筑.

姜子炎(1979－), 男, 博士, 助理研究员, 主要研究领域为建筑自动化, 智能控制.

邢建春(1964－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为物联网, 国防工程智能化.

周启臻(1993－), 男, 博士, 讲师, 主要研究领域为物联网, 无线感知.

邹荣伟(1996－), 女, 博士生, 主要研究领域为软件工程, 智能建筑.

冯博伟(1996－), 男, 博士生, 主要研究领域为智能建筑, 无线感知.

© 中国科学院软件研究所 http://www.jos.org.cn