线下方 LS 的消息处理循环,LS 接到 *VICTIM* 消息后要判断本地任务队列里有没有多的任务:若有,则从队列尾部取出一个任务打包发送给请求节点;若无,则不用专门通知请求节点,因为马上该节点就会向 GS 发送工作窃取请求,而此时,该节点应是任务数最多的节点,因此,GS 会很快判断出所有任务结束,并给各个节点发送计算终止消息.原则上,出错队列中的任务如果在本节点内被所有 PE 对执行后还不能得到一致结果,则应迁移到其他节点执行,但在目前 FT-TPP 的实现中,为使队列管理等不过于复杂,我们不允许出错任务在节点间迁移.

4 实验与分析

4.1 实验平台与测试程序

为测试 FT-TPP 的容错能力和对程序性能的影响,我们基于 FT-TPP 分别实现了表 1 中的应用,这些应用涵盖了 FT-TPP 支持的 3 种任务并行模式,Fib,Nq 和 Ms 属于递归式并行,MM 属于水平式并行,St 本身可实现为 3 种模式中的任意一种.这里,我们对 Strassen 算法递归几次后形成的任务 DAG 实现为不规则并行模式.

实验在一个 16 节点的多核集群上进行,各节点基本配置为:2.4GHz Intel Xeon E5620 处理器(支持 8 个硬件线程),12G 内存,Linux 内核 3.4.

Table 1 Benchmark applications

表 1 测试程序

名称	描述
Fib (*n*)	递归地计算斐波那契数列的第 *n* 项值
Nq (*n*)	求解 *n* 皇后问题
MM (*n*)	标准矩阵乘法的并行实现,对最外层循环并行化(*n*×*n* double matrix)
Ms (*n*)	对长度为 *n* 的整数类型数据进行并行归并排序
St (*n*)	Strassen 快速矩阵乘法(*n*×*n* double matrix)[37]

4.2 容错性能分析

为检验 FT-TPP 在瞬时错误检测与恢复方面的效果,我们用 Pin 工具[38]在测试用例运行过程中注入 SEU(单位翻转)错误,错误注入后的程序行为可分为 4 类:(1) 检测到该瞬时错误并进行错误恢复(detected);(2) 发生段错误退出程序(seg fault);(3) 程序运行超时(timeout),亦即受错误影响的线程在所有任务都结束后始终不能退出,可能陷入了一个死循环;(4) 程序正常结束并输出正确结果(benign).每个测试程序各运行 1 000 次,为控制程序运行时间,我们在本实验中采用较小的输入参数,实验统计上述 4 种程序行为发生的比例,结果如图 10 所示.

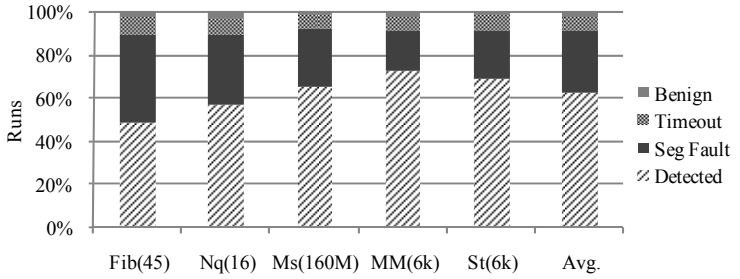

Fig.10 Fault detection distribution of FT-TPP with transient faults

图 10 FT-TPP 瞬时错误检测与恢复情况分布

平均来看,有 62%的错误被检测和恢复,28%的错误引起段错误,还有 6.8%的超时错误.另外,虽然有 38%的错误未被检测到,但由于 FT-TPP 的动态任务迁移和永久错误检测与恢复机制,所有程序最终都成功完成了计算任务.从这一点来看,FT-TPP 实际提供了 100%的错误覆盖率.但在理论上,FT-TPP 的错误覆盖率不可能是 100%,因为程序可能在下述两种发生概率极低的情况下输出错误结果:(1) 两次瞬时错误发生在同一任务两次执行过程中的相同位置;(2) 错误发生在所有任务队列都为空且出错任务队列里的任务最后一次直接执行并提交的过

程中(见第 3.3.1 节).

为检验 FT-TPP 在永久错误容忍方面的效果,我们在程序运行过程中通过 kill 相关计算进程来模拟节点故障,我们进行了两次实验:第 1 次实验在每个程序运行过程中随机终止一个节点上的计算进程,第 2 次随机终止两个节点上的计算进程.实验中,每个应用分别执行 100 次并计算平均执行时间,结果如图 11 所示.为便于比较,各执行时间以无错误情况下的执行时间为标准归一化,结果可见:单节点故障情况下程序执行时间平均比无故障情况增加了 2.76%,双节点故障情况下程序执行时间平均增加了 5.6%.总的来看,FT-TPP 能很快恢复故障节点上的任务,并成功完成所有计算任务.这归功于 FT-TPP 所采用的无盘检查点机制和动态任务调度,故障节点上未完成的任务从检查点恢复由调度器分配给各处理单元并行执行.

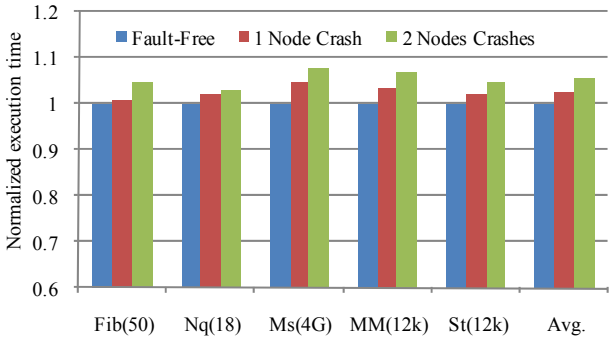

Fig.11 Performance of FT-TPP with permanent faults

图 11 FT-TPP 永久错误检测与恢复的性能

4.3 开销分析

为评估 FT-TPP 的容错开销,我们将其实现中与容错相关的部分剥离出去,形成一个普通的不支持容错的任务并行程序设计模型,简记为 NoFT-TPP,比较 FT-TPP 与 NoFT-TPP 就可对 FT-TPP 的容错开销有一个总体了解.NoFT-TPP 实际上与文献[39]工作类似,是一个层次化工作窃取任务调度框架.我们将表 1 中的应用基于 FT-TPP 和 NoFT-TPP 分别实现,对每个测试程序各执行 40 次并计算平均执行时间,运行过程中不引入任何错误,结果如图 12 所示.

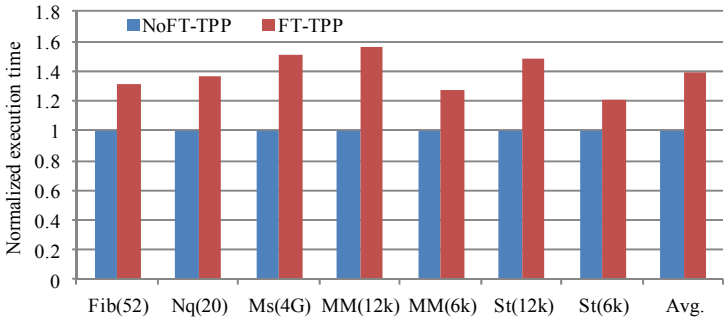

Fig.12 Performance comparison of FT-TPP with NoFT-TPP

图 12 FT-TPP 与 NoFT-TPP 的性能比较

各执行时间以 NoFT-TPP 为标准归一化,与 NoFT-TPP 相比,FT-TPP 执行时间平均增加了 39%.也就是说,FT-TPP 总体容错开销大约为 39%.这些开销主要来自于 Buffer-Commit、检查点相关操作、冗余执行等几个方面.

Buffer-Commit 和检查点的开销与任务输出的数据量相关,对于 Fib(n)和 Nq(n),每个任务仅输出少量数据,而对 MM(n),Ms(n)和 St(n)而言,每个任务会产生大量数据,因此在图 12 中,Fib(52)和 Nq(20)的 FT-TPP 容错开销低于 Ms(4G),MM(12k)和 St(12k).FT-TPP 对每个任务要执行 2 次,而 NoFT-TPP 对每个任务只执行 1 次,因此理

论上,FT-TPP 的容错开销应高于 100%,但在图 12 中,FT-TPP 的最大容错开销仅为 57%,见 MM(12k).这主要归功于 FT-TPP 对任务之间并行性的充分发掘,在处理单元数量充足的情况下,冗余任务也被均衡地调度到各个处理单元上并行执行,细粒度的任务并行和高效的负载均衡策略使得执行时间并不加倍.另外,比较图 12 中 MM(12k)与 MM(6k),St(12k)与 St(6k)的执行时间可知:数据量的减少能显著降低 FT-TPP 的容错开销,St(6k)的容错开销只有 21%,这与 St(12k)(48%)相比降低了超过一半.这是因为,我们在实验中观察到:对较小的数据量,在 MM(n)和 St(n)的执行过程中,有的处理单元由于得不到任务,长时间处于空闲状态,FT-TPP 的冗余任务执行使其资源利用率在这种情况下高于 NoFT-TPP.

为进一步分析 FT-TPP 瞬时错误检测的性能开销,即,本文 Buffer-Commit 计算模型与容错工作窃取调度策略的效果,我们将 FT-TPP 和 NoFT-TPP 简化为共享存储模式,即,去除其中节点间消息传递相关部分.另外,对 FT-TPP 去除检查点相关操作,然后在实验平台的一个计算节点上运行各测试程序,比较简化后的 FT-TPP 与 NoFT-TPP,分别记为 FT-TPP-S 与 NoFT-TPP-S(S 表示针对共享存储).结果如图 13 所示:平均来看,FT-TPP 以 27.4%的性能开销提供了瞬时错误的检测能力.

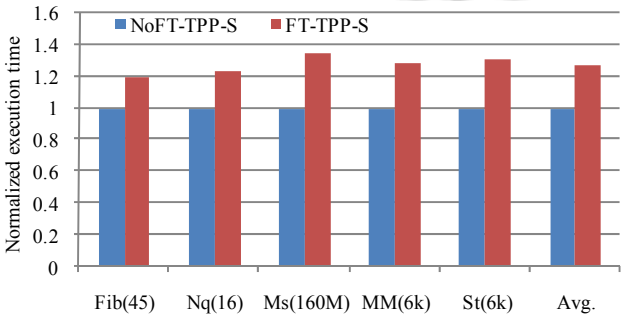

Fig.13 Performance comparison of FT-TPP-S with NoFT-TPP-S

图 13 FT-TPP-S 与 NoFT-TPP-S 的性能比较

5 总 结

本文提出了一种支持容错的任务并行程序设计模型 FT-TPP,FT-TPP 在发掘任务级并行性以提高系统性能的同时,实现了任务粒度的错误检测和恢复机制,以保证系统可靠性.FT-TPP 目前支持水平、递归和不规则并行这 3 种并行模式,调度策略采用分层工作窃取任务调度,主要采用以下关键技术实现容错支持:Buffer-Commit 的计算模型、容错工作窃取任务调度以及应用级无盘检查点.

进一步的工作将围绕以下几个方面进行:(1) 考虑对流水并行模式的支持;(2) 探讨编译器辅助进行的检查点任务数据生成方法;(3) 优化运行时库的实现减小实际性能开销;(4) 将本文容错工作窃取任务调度等技术应用于 TBB 等现有并行程序设计语言和运行库中.

References:

[1] Reinders J. Intel Threading Building Blocks. 2015. https://www.amazon.com/Intel-Threading-Building-Blocks-Parallelism/dp/0596514808

[2] Charles P, Grothoff C, Saraswat V, Donawa C, Kielstra A, Ebcioglu K, von Praun C, Sarkar V. X10: An object-oriented approach to non-uniform cluster computing. In: Johnson R, ed. Proc. of the 20th Annual ACM SIGPLAN Conf. on Object-Oriented Programming. New York: ACM Press, 2005. 519−538. [doi: 10.1145/1103845.1094852]

[3] Frigo M, Leiserson CE, Randall KH. The implementation of the cilk-5 multithreaded language. In: Berman AM, ed. Proc. of the ACM SIGPLAN'98 Conf. on Programming Language Design and Implementation. New York: ACM Press, 1998. 212−223. [doi: 10.1145/277650.277725]

[4] Wang L, Cui HM, Chen L, Feng XB. Research on task parallel programming model. Ruan Jian Xue Bao/Journal of Software, 2013, 24(1):77−90 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[5] Leijen D, Schulte W, Burckhardt S. The design of a task parallel library. In: Arora S, ed. Proc. of the 24th Annual ACM SIGPLAN Conf. on Object-Oriented Programming. New York: ACM Press, 2009. 227−242. [doi: 10.1145/1640089.1640106]

[6] Yang C, Orailoglu A. Full fault resilience and relaxed synchronization requirements at the cache-memory interface. IEEE Trans. on VLSI System, 2011,19(11):1996−2009. [doi: 10.1109/TVLSI.2010.2067230]

[7] Reinhardt SK, Mukherjee SS. Transient fault detection via simultaneous multithreading. In: Berenbaum A, ed. Proc. of the 27th Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 2000. 25−36. [doi: 10.1145/339647.339652]

[8] Oh N, Shirvani PP, McCluskey EJ. Error detection by duplicated instructions in super-scalar processors. IEEE Trans. on Reliability, 2002,51(1):63−75. [doi: 10.1109/24.994913]

[9] Reis GA, Chang J, Vachharajani N, Rangan R, August DI. SWIFT: Software implemented fault tolerance. In: Conte T, ed. Proc. of the Int'l Symp. on Code Generation and Optimization. Washington: IEEE Computer Society, 2005. 243−254. [doi: 10.1109/CGO.2005.34]

[10] Rotenberg E. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In: Proc. of the 29th Annual Int'l Symp. on Fault-Tolerant Computing. Madison: IEEE, 1999. 84−91. [doi: 10.1109/FTCS.1999.781037]

[11] Mukherjee SS, Kontz M, Reinhardt SK. Detailed design and evaluation of redundant multithreading alternatives. In: Skadron K, ed. Proc. of the 29th Annual Int'l Symp. on Computer Architecture. Washington: IEEE Computer Society, 2002. 99−110. [doi: 10.1109/ISCA.2002.1003566]

[12] Shye A, Blomstedt J, Moseley T, Reddi VJ, Connors DA. PLR: A software approach to transient fault tolerance for multicore architectures. IEEE Trans. on Dependable and Secure Computing, 2009,6(2):135−148. [doi: 10.1109/TDSC.2008.62]

[13] Randell B, Lee P, Treleaven PC. Reliability issues in computing system design. ACM Computing Surveys, 1978,10(2):123−165. [doi: 10.1145/356725.356729]

[14] Du YF. The study and analysis on fault-tolerant parallel algorithm [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2008 (in Chinese with English abstract).

[15] Bronevetsky G, Marques D, Pingali K, Szwed PK, Schulz M. Application-Level checkpointing for shared memory programs. In: Proc. of the 11th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2004. 235−247. [doi: 10.1145/1024393.1024421]

[16] Chen Z. Adaptive checkpointing. Journal of Communications, 2010,5(1):81−87. [doi: 10.4304/jcm.5.1.81-87]

[17] Huang KH, Abraham JA. Algorithm-Based fault tolerance for matrix operations. IEEE Trans. on Computers, 1984,33(6):518−528. [doi: 10.1109/TC.1984.1676475]

[18] Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1):107−113. [doi: 10.1145/1327452.1327492]

[19] NVIDIA Corp. NVIDIA CUDA. 2010. http://developer.nvidia.com/object/cuda.html/

[20] Khronos. OpenCL: The open standard for parallel programming of heterogeneous systems. 2010. http://www.khronos.org/opencl/

[21] Sophisticated Library for Vector Parallelism. Intel array building blocks. 2010. http://software.intel.com/en-us/articles/intel-array-building-blocks/

[22] El-Ghazawi TA, Carlson WW, Draper JM. UPC Language Specification v1.1.1. 2003.

[23] Cray Inc. The chapel language specification version 0.4. Technical Report, Cray Inc., 2005.

[24] Tannenbaum T, Jim B, Litzkow M, Livny M. Checkpoint and migration of Unix processes in the condor distributed processing system. Technical Report, 1346, University of Wisconsin-Madison Computer Sciences, 1997.

[25] Burns G, Daoud R, Vaigl J. LAM: An open cluster environment for MPI. In: Proc. of the Supercomputing Symp. Ontario: University of Toronto, 1994. 379−386.

[26] Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain RH, Daniel DJ, Graham RL, Woodall TS. Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Kranzlmuller D, ed. Proc. of the 11th European PVM/MPI Users' Group Meeting. Berlin: Springer-Verlag, 2004. 97−104. [doi: 10.1007/978-3-540-30218-6_19]

[27] Fagg GE, Gabriel E, Chen Z, Angskun T, Bosilca G, Pjesivac-Grbovic J, Dongarra JJ. Process fault-tolerance: Semantics, design and applications for high performance computing. Int'l Journal of High Performance Computing Applications, 2005,19(4):465−477. [doi: 10.1177/1094342005056137]

[28] Blumofe RD. Executing multithreaded programs efficiently [Ph.D. Thesis]. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1995.

[29] van Nieuwpoort RV, Wrzesińska G, Jacobs CJH, Bal HE. Satin: A high-level and efficient grid programming model. ACM Trans. on Programming Languages and Systems, 2010,32(3):39. [doi: 10.1145/1709093.1709096]

[30] Baldeschwieler JE, Blumofe RD, Brewer EA. Atlas: An infrastructure for global computing. In: Proc. of the 7th Workshop on ACMSIGOPS European Workshop. New York: ATLAS, 1996. 165–172. http://dl.acm.org/citation.cfm?id=504482&dl=ACM&coll=DL&CFID=632100309&CFTOKEN=67677564

[31] Taura K, Kaneda K, Endo T, Yonezawa A. Phoenix: A parallel programming model for accommodating dynamically joining/leaving resources. In: Eigenmann R, ed. Proc. of the 9th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2003. 216–229. [doi: 10.1145/781498.781533]

[32] Bahi JM, Hakem M, Mazouzi K. Reliable parallel programming model for distributed computing environments. In: Lin HX, ed. Proc. of the Euro-Par Workshops. Berlin: Springer-Verlag, 2010. 162–171. [doi: 10.1007/978-3-642-14122-5_20]

[33] Flynn-Hummel S, Schonberg E, Flynn LE. Factoring: A method for scheduling parallel loops. Communications of the ACM, 1992, 35(8):90–101. [doi: 10.1145/135226.135232]

[34] Blumofe RD, Leiserson CE. Scheduling multithreaded computations by work stealing. Journal of the ACM, 1999,46(5):720–748. [doi: 10.1145/324133.324234]

[35] Zhang W, Kruijf M, Li A, Lu S, Sankaralingam K. ConAir: Featherweight concurrency bug recovery via single-threaded idempotent execution. In: Sarkar V, ed. Proc. of the 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2013. 113–126. [doi: 10.1145/2451116.2451129]

[36] Feng M, Gupta R, Hu Y. SpiceC: Scalable parallelism via implicit copying and explicit commit. In: Cascaval C, ed. Proc. of the 16th ACM Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2011. 69–80. [doi: 10.1145/2038037.1941564]

[37] Strassen V. Gaussian elimination is not optimal. Numerische Mathematik, 1969,14(3):354–356.

[38] Luk CK, Lowney PG, Hazelwood KM. Pin: Building customized program analysis tools with dynamic instrumentation. In: Sarkar V, ed. Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 2005. 190–200. [doi: 10.1145/1065010.1065034]

[39] Quintin JN, Wagner F. Hierarchical work-stealing. In: D'Ambra P, ed. Proc. of the 16th Int'l Euro-Par Conf. Berlin: Springer-Verlag, 2010. 217–229. [doi: 10.1007/978-3-642-15277-1_21]

附中文参考文献:

[4] 王蕾,崔慧敏,陈莉,冯晓兵.任务并行编程模型研究与进展.软件学报,2013,24(1):77–90. http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[14] 杜云飞.容错并行算法的研究与分析[博士学位论文].长沙:国防科学技术大学,2008.

王一拙(1979－),男,陕西西安人,博士,讲师,CCF 会员,主要研究领域为计算机体系结构,并行程序设计.

苏岩(1980－),男,博士生,主要研究领域为计算机体系结构.

陈旭(1983－),男,博士生,主要研究领域为计算机体系结构.

王小军(1979－),男,讲师,主要研究领域为计算机体系结构,嵌入式系统.

计卫星(1980－),男,博士,副教授,CCF 会员,主要研究领域为计算机体系结构,并行计算,嵌入式系统.

石峰(1961－),男,博士,教授,博士生导师,主要研究领域为计算机体系结构,嵌入式系统.