A. 任务划分

在并行交叉时,与并行 Fitness 相似,同样可以利用此时个体与个体间相关性不大的特性,使用一个 Block 处理一个个体.此时,同样地,不同的 Block 之间不需要共享数据,可以达到数据独立.而在一个 Block 内部,需要考虑如何按照基因位置的映射生成子代的基因.此时,采用一个 Thread 处理一个基因位的方式来生成子代的基因.一个 Block 内同样采用了 512 个 Thread,每个 Thread 可以独立处理一个基因位,即如果种群包含 P 个个体,每个个体有 N 个基因位,则其并行化算法可以通过使用 $P\times512$ 个线程进行计算(如果 $N<512$,则可使用 $P\times N$ 个线程计算).

与并行计算 Fitness 相同,每个 Block 中的线程在计算过程需要同步,单个 Block 计算中耗时最长的线程决定了整个计算过程的耗时.由于 Block 中的计算用了 512 为一组的策略,整个并行交叉的算法复杂度为$\lceil M/512\rceil$.

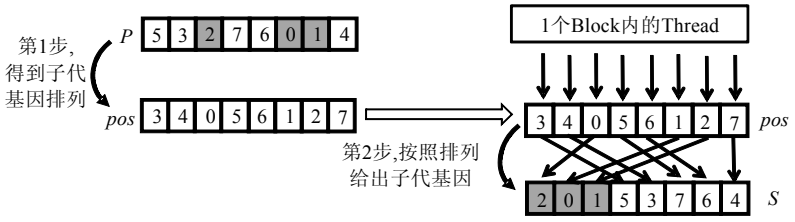
B. 数据传输

在串行算法得到所有的 *pos* 向量之后,产生子代需要父代的基因以及 *pos* 向量,传入数据量为 $P\times N$;而并行策略中对于父代信息的访问是通过 id 与种群矩阵的方式,即需要传入 $P\times N+P$.因此,最终的传入数据量为 $2P\times N+P$.在产生了子代之后,需要将子代的测试用例序列传出,传出数据量为 $P\times N$.

C. 数据存储

在交叉并行过程中,由于子代与父代的测试用例序列信息会被所有的 Block 访问,而该信息本身较大,因此将其存储于 GPU 的 Global Memory 中.在组成子代的测试用例序列过程中,由于父代的 id 会经常被访问,因此采用了读取速度最高的 Cache 进行存储.

针对以上单点交叉算法,具体并行策略如图 4 所示,其中,图 4(a)所示为通过串行计算得到子代基因的新位置.例如,5 号基因对应的 *pos* 中的数值为 3,表示 5 号基因在子代中的新位置为 3(位置从 0 开始计算),以此类推.图 4(b)所示为一个 Block 内的 8 个 Thread 分别处理 8 个位置上的基因,将原有基因按照 *pos* 的指导放在对应的位置上,最终得到子代的基因序列.在并行交叉过程中,由于个体中每个基因位交叉后的位置为输入,因此每个 Block 在处理一个个体内的基因时是相互独立的,即,GPU 只需要将基因按照 *pos* 的映射生成子代的基因.



(a) Crossover 第 1 步(串行) (b) Crossover 第 2 步(并行)

Fig.4 Parallel Crossover strategy

图 4 Crossover 并行策略

3 实验及结果

为验证本文提出的基于CPU+GPU异构模式下的多目标测试用例优先排序技术,设计并计算不同并行计算模式在效率上的差异,我们进行了一系列的实验,在不同规模软件的测试用例优先排序过程中,度量了不同并行模式下 NSGA-II 算法各个操作执行效率的差异.

3.1 实验环境

表 1 给出了实验中所用到的被测程序及其代码有效行数和测试用例集的大小,其中,v8 为 Google 浏览器 Chrome 的 JavaScript 引擎,属开源项目,其余 3 个都来自 SIR 测试用例库中程序规模相对较大的程序.为了获取测试用例集,用随机算法从测试用例池中抽取测试用例,直至达到最大覆盖.针对 flex,space,bash 实验程序共抽取了 100 组测试用例集;针对 v8 抽取了 30 组测试用例集.表 1 中,最后 3 行给出了同一被测程序所有测试用例

集所包含的测试用例的最大、最小和平均个数.实验平台为一台 GPU 服务器,配有 16 核英特尔 Xeon(R) CPU E5620 主频 2.40GHz,并行平台所使用的显卡为 Nvidia 公司的一块 Tesla M2050,CUDA 版本为 5.5,编程语言为 C++.

通过 Nvidia SDK 自带的 Benchmark 程序测出实验平台数据传输速率具体为:从内存到显存的带宽为 2 961.1MB/s,从显存到内存的带宽为 3 198.8MB/s.遗传算法种群大小设定为 64,交叉概率与变异概率为 100%,迭代次数为 100.

Table 1 The statistics of software under test

表 1 被测程序及其规模

被测程序		flex	space	bash	v8
代码有效行		3 016	3 815	6 181	59 412
测试用例集	最小	1 047	1 208	764	2 454
	最大	1 470	3 229	1 467	5 585
	平均	1 350.17	1 894.29	1 063.17	3 737.23

3.2 实验设计

在基于 CPU+GPU 的异构并行多目标测试用例优先排序中,我们设计了 Fitness 和 Crossover 的并行策略实验,分别进行了完全串行、单独并行 Fitness、单独并行 Crossover 以及两者同时并行的 4 组实验.为保证实验的有效性,针对 flex,space,bash 的每个集合,实验重复执行了 30 次;针对 v8,由于程序规模巨大,每个实验只重复了 10 次.实验结果取平均值.

3.3 实验结果

在实验中,我们统计了完全串行、单独并行 Fitness、单独并行 Crossover 以及两者同时并行时的总体运行时间和一次迭代中所执行的各个操作所需要的时间.在本节中,我们以表格的形式列出所有实验数据,在下一节中,我们将详细分析实验结果.

表 2 列出了完全串行的实验结果:所需总体时间以及各个操作所需要的平均时间.迭代平均时间是指迭代一次各个操作所消耗的时间和的平均值,而总时间为遗传算法迭代 100 次所消耗的时间.其中,初始化操作包含了覆盖信息的读取、一些变量的初始化以及首次的 Fitness 计算.从表 2 可以看出:Fitness 计算所消耗的时间占到了每次迭代过程时间的绝大部分,其次是交叉操作的时间.这个数据显示:本文的并行策略设计是正确的,把串行中最耗时的部分并行化,理论上可以极大地提高效率.

Table 2 Time consumption of serial process (ms)

表 2 串行操作所需时间(单位:毫秒)

程序	初始化	选择时间	交叉时间	变异时间	Fitness 计算	精英选择	迭代平均时间	总时间
flex	817.08	0.07	40.01	0.02	461.30	6.49	507.82	51 598.81
space	4 486.93	0.07	55.32	0.02	722.38	7.11	784.83	82 969.75
bash	2 053.04	0.07	28.40	0.02	1 632.39	5.57	1 666.38	168 690.93
v8	64 423.36	0.07	99.67	0.02	44 794.12	8.39	44 902.20	4 574 940.76

表 3~表 5 分别给出了单独并行 Fitness、单独并行 Crossover 以及两者同时并行时各个操作的具体时间消耗.注意:由于并行产生了数据传输,不同的并行策略中数据传输的次数不同.在表 3~表 5 中,同时统计了其产生的时间消耗.

Table 3 Time consumption of different operators in parallel Fitness process (ms)

表 3 并行 Fitness 时,各个操作的时间消耗(单位:毫秒)

程序	初始化	选择时间	交叉时间	变异时间	传入时间	Fitness 计算	传出时间	精英选择	迭代平均时间	总时间
flex	805.11	0.08	44.43	0.02	3.88	14.25	4.31	7.66	129.51	13 756.37
space	1 095.03	0.08	58.80	0.02	2.20	50.09	0.98	7.97	202.44	21 339.20
bash	945.86	0.08	33.27	0.02	1.42	25.15	9.05	6.88	154.55	16 400.95
v8	13 427.14	0.08	122.53	0.03	4.92	547.99	3.20	10.72	1501.99	163 626.60

Table 4 Time consumption of different operators in parallel Crossover process (ms)

表 4 并行 Crossover 时,各个操作的时间消耗(单位:毫秒)

程序	初始化	选择时间	传入时间	交叉时间	传出时间	变异时间	Fitness计算	精英选择	迭代平均时间	总时间
flex	899.65	0.08	0.47	20.69	0.39	0.02	458.47	6.64	486.74	49 573.84
space	4 377.85	0.07	0.63	28.87	0.53	0.02	673.76	7.56	711.44	75 521.51
bash	2 221.37	0.07	0.34	15.42	0.34	0.02	1 677.34	6.57	1 700.11	172 231.87
v8	899.65	0.07	1.09	54.37	0.92	0.02	51 280.87	18.84	51 356.17	5 203 800.74

Table 5 Time consumption of different operators in parallel Fitness & Crossover process (ms)

表 5 同时并行 Fitness 与 Crossover 时,各个操作的时间消耗(单位:毫秒)

程序	初始化	选择时间	交叉传入时间	交叉时间	交叉传出时间	变异时间	Fitness传入时间	Fitness计算	Fitness传出时间	精英选择	迭代平均时间	总时间
flex	863.57	0.08	0.47	24.94	2.87	0.02	0.65	15.25	1.29	8.55	88.66	9 730.02
space	1 215.92	0.08	0.63	34.63	0.97	0.02	0.64	49.57	0.53	9.70	158.25	17 041.35
bash	983.48	0.08	0.34	19.19	4.23	0.02	0.46	25.81	5.56	8.66	123.73	13 356.08
v8	12 086.06	0.08	1.13	68.63	1.13	0.02	2.20	604.11	0.75	22.03	1 438.92	155 978.49

4 实验结果分析

本文提出了基于 CPU+GPU 的异构并行多目标测试用例优先排序,所以,并行所带来的整体效率提升是我们首要关注的问题;其次,我们设计了多目标优化算法 NSGA-II 中 Fitness 和 Crossover 的并行策略,不同的并行策略对整体效率提升的效果是实验分析的另一个重点;最后,实验中采用了具有不同测试用例集大小的不同规模的程序,影响并行效率提升的相关因素分析也是实际中应用并行要考虑的.

4.1 整体效率提升分析

根据表 5 的数据,我们计算了串行总体时间以及同时并行 Fitness 和 Crossover 时所花费的总体时间的比值,即加速比,如图 5 所示.可以看出:在针对程序规模较小的 flex 与 space 时(有效代码行 3 000 左右),加速比在 4 左右;在针对 bash 的实验中,加速比达到了 10 以上;而在针对 v8 的实验中,程序规模增加到近 6 万行有效代码,加速比接近 30.再进一步通过表 5 数据可以看出:v8 程序在本文提出的基于 CPU+GPU 异构并行模式下执行多目标测试用例优先排序,可以在不到 3 分钟内(160 秒内)完成排序,这个实验结果已经可以被工业界所接受.

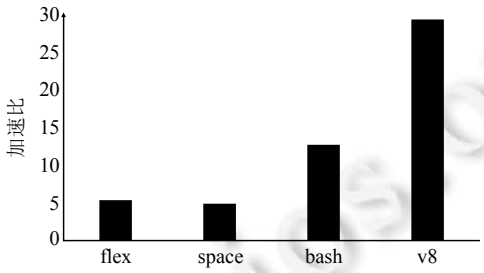


Fig.5 Speed-Up rates of parallel Fitness and Crossover process

图 5 同时并行 Fitness 与 Crossover 的整体加速比

4.2 不同并行策略的比较

从表 2 的数据中可以看出:Fitness 计算占总体时间的绝大部分,而 Crossover 是所有遗传操作中时间花费最大的.所以在 Fitness 并行的基础上,进一步设计了 Crossover 的并行.图 6 根据表 3 和表 5 的数据,计算了单独并行 Fitness 和同时并行 Fitness+Crossover 的加速比.可以明显看出:针对 4 个被测程序,后者比前者加速比略高.虽然两者差距不大,但可以看出,差距随着程序规模的扩大而增大.所以,在 Fitness 并行的基础上增加 Crossover

的并行,可以进一步提高计算效率.随着程序规模的扩大,效果增加得越显著.

为了进一步分析并行交叉的效率提升情况,图 7 给出了 4 个程序单独并行交叉相对于串行交叉的加速比.可以看出:总体加速比在 1.6~1.8 之间,其中,v8 程序规模最大,单独并行交叉的加速比反而最小.与 Fitness 并行不同,并行交叉操作的加速比随程序规模大小的变化不够明显.具体原因如第 2.2 节所述:为了保证交叉算子并行策略的通用性,将交叉操作分为两步,第 1 步串行执行,仅第 2 步并行执行,所以加速比不大.综合考虑表 4 中的交叉时间,只占单次迭代时间的 5%左右,提升并行交叉算子的加速比对整体加速比影响不大.

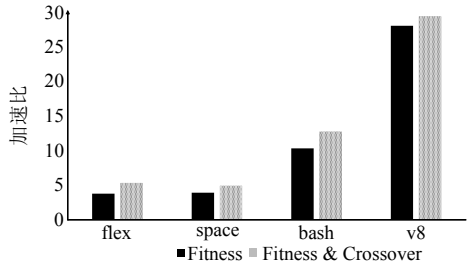


Fig.6 Comparison of speed-up rates beteeen Fitness parallellization and Fitness & Crossover parallelization

图 6 单独并行 Fitness 和同时并行 Fitness 与 Crossover 的整体加速比

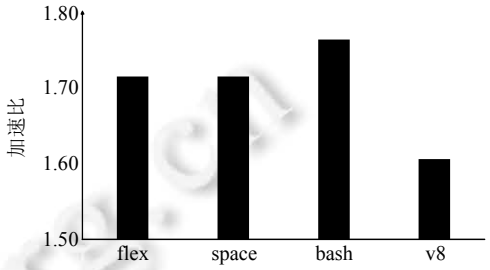


Fig.7 Speed-up rates of parallel Crossover process over serial Crossover process

图 7 并行 Crossover 相对串行 Crossover 的加速比

综合以上结果可以得到:针对 Fitness 计算并行加速的整体收益非常大;而在此基础上增加交叉的并行,其整体收益有增加但并不明显.同时,从第 2.2 节中可以看到:由于交叉操作的复杂性,在实现 GPU 并行编程时的算法难度较大.因此,建议在 CPU+GPU 异构模式下,测试用例排序并行计算可以只考虑单独并行 Fitness 计算.

4.3 算法加速比影响因素分析

在 CPU+GPU 异构并行计算中,数据要在 CPU 和 GPU 进行交换.为了分析数据交换对并行加速比的影响,表 3~表 5 给出了数据传入时间与传出时间,可以发现:并行交叉算子的数据传输中,数据的传入与传出时间基本上均小于 1ms;在并行 Fitness 计算的数据传输中,针对被测程序为 bash 与 v8,由于程序规模和测试用例集规模均较大,所以耗时较大,但是传入时间基本上在 5ms 左右,数据交换的耗时占整体运行时间的比例不大,对于整个加速比的影响几乎可以忽略.从图 6 中可以看出,并行加速比基本与程序规模成正比.即:随着程序规模的扩大,并行加速比随之增大.所以在回归测试中,程序规模越大,采用测试用例优先排序技术的时间花费越大,采用 GPU 并行的效率提升情况越明显.

由于 Fitness 计算占整体时间的绝大部分,我们进一步分析了单独并行 Fitness 的加速比.由表 3 可以得出,程序针对 Fitness 的加速比与被测程序规模成正相关.这是由于,Fitness 计算需要遍历测试用例的覆盖矩阵,而 TS_i 的计算次数等于被测程序的代码行数.一次计算的 TS_i 次数越多,并行实现所带来的额外开销在总时间中所占比重就越低,图 6 中计算并行的加速比就越大.可以注意到,space 程序加速比略微偏小可能与测试用例集相对较大有关.

4.4 实验结果影响因素分析

在本文实验中的 Fitness 采用了平均语句覆盖 APSC,实际应用中可以采用平均分支覆盖(APDC)或平均语句块覆盖(APBC).后两种从覆盖矩阵规模上远远小于 APSC,但计算公式相似,可以直接采用本文的并行策略.为了更好地验证规模对并行效率提升的影响,本文采用了 APSC.

SIR 库提供了十几个开源程序和测试用例集合,本文实验中只选择了 3 个规模较大的程序,同时补充了 1 个工业界的开源程序 v8,程序规模是 SIR 库中最大规模程序的 10 倍,可以有效地分析程序规模和并行加速效率提升之间的关系.为保证实验的有效性,本文采用了 SIR 抽取测试用例集的方法,对 4 个程序重新抽取了测试用例

集合,保证了程序之间的一致性.

由于本文所设计的实验较多,如果单独运行需要花费近 1 年的时间,因此大部分实验采取了在同一台服务器上多线程并行运行的方式,会给实验结果带来一定的误差.但经过我们的分析,实验所带来的误差并不影响最终结论的得出.

5 总 结

本文针对软件回归测试用例优先排序的效率问题,提出了面向 CPU+GPU 异构计算的多目标测试用例优先排序,并就不同的异构并行策略进行研究分析.实验测试了 4 个规模较大的被测程序,研究了 NSGA-II 算法中序列编码的 Fitness 并行策略和 Crossover 并行策略,并根据实验结果分析,提出一个可行 CPU+GPU 异构架构下并行模式.实验结果显示:针对近 6 万行有效代码的工业界开源程序 v8,测试用例集的大小接近 4 000,在本文提出的 CPU+GPU 异构并行框架下,单 GPU 就实现了近 30 倍的加速提升,可以在 3 分钟内完成测试用例的多目标优先排序,可以有效地推动测试用例优先排序技术在工业界的实际应用.

References:

[1] Jiang B, Chan W. On the integration of test adequacy, test case prioritization, and statistical fault localization. In: Proc. of the 10th Int'l Conf. on Quality Software (QSIC). IEEE, 2010. 377−384. [doi: 10.1109/QSIC.2010.64]

[2] Zhang XF, Xu BW, Nie CH, Shi L. Approach for optimizing test suite based on testing requirement reduction. Ruan Jian Xue Bao/Journal of Software, 2007,18(4):821−831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20070404.htm

[3] Gejian D, Yanni Z, Lu Z. Study of test suite minimization based on ant colony algorithm. Computer Engineering, 2009,35(6): 213−218 (in Chinese with English abstract).

[4] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verication and Reliability, 2012,22(2):67−120. [doi: 10.1002/stv.430]

[5] Wong WE, Horgan JR, London S, Mathur AP. Effect of test set minimization on fault detection effectiveness. In: Proc. of the 17th Int'l Conf. on Software Engineering (ICSE'95). IEEE, 1995. 41. [doi: 10.1145/225014.225018]

[6] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

[7] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33:225−237. [doi: 10.1109/TSE.2007.38]

[8] Hla KHS, Choi YS, Park JS. Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In: Proc. of the IEEE 8th Int'l Conf. on Computer and Information Technology Workshops (CIT Workshops 2008). IEEE, 2008. 527−532. [doi: 10.1109/CIT.2008.Workshops.104]

[9] Singh Y, Kaur A, Suri B. Test case prioritization using ant colony optimization. ACM SIGSOFT Software Engineering Notes, 2010, 35(4):1−7. [doi: 10.1145/1811226.1811238]

[10] McMinn P. Search-Based software testing: Past, present and future. In: Proc. of the IEEE 4th Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2011. 153−163. [doi: 10.1109/ICSTW.2011.100]

[11] Harman M, McMinn P, de Souza JT, Yoo S. Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification. LNCS 7007, 2012. 1−59. http://www0.cs.ucl.ac.uk/staff/mhaman/laser.pdf

[12] Srinivas N, Deb K. Multi-Objective function optimization using non-dominated sorting genetic algorithms. IEEE Trans. on Evolutionary Computation, 1994,2(3):221−248. [doi: 10.1162/evco.1994.2.3.221]

[13] Li H, Zhang Q. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. IEEE Trans. on Evolutionary Computation, 2009,13(2):284−302. [doi: 10.1109/TEVC.2008.925798]

[14] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2000,1917:849−858. [doi: 10.1109/4235.996017]

[15] Nucci DD, Panichella A, Zaidman A, Lucia AD. Hypervolume-Based search for test case prioritization. Lecture Notes in Computer Science, 2015,9275:157−172. [doi: 10.1007/978-3-319-22183-0_11]

[16] Yuan F, Bian Y, Li Z, Zhao R. Epistatic genetic algorithm for test case prioritization. In: Proc. of the Search-Based Software Engineering. Springer-Verlag, 2015. 109−124. [doi: 10.1007/978-3-319-22183-0_8]

[17] Srinivas M, Patnaik LM. Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans. on Systems, Man and Cybernetics, 1994,24(4):656−667. [doi: 10.1109/21.286385]

[18] Yoo S, Harman M, Ur S. Highly scalable multi objective test suite minimisation using graphics cards. In: Proc. of the Search Based Software Engineering. Springer-Verlag, 2011. 219−236. [doi: 10.1007/978-3-642-23716-4_20]

[19] Epitropakis MG, Yoo S, Harman M, Burke EK. Empirical evaluation of Pareto efficient multi-objective regression test case prioritization. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. ACM, 2015. 234−245.

[20] Walcott KR, Soffa ML, Kapfhammer GM, Roos RS. Timeaware test suite prioritization. In: Proc. of the 2006 Int'l Symp. on Software Testing and Analysis. ACM Press, 2006. 1−12. [doi: 10.1145/1146238.1146240]

附中文参考文献:

[2] 章晓芳,徐宝文,聂长海,史亮.一种基于测试需求约简的测试用例集优化方法.软件学报,2007,18(4):821−831. http://www.jos.org.cn/1000-9825/20070404.htm

[3] 丁革建,郑燕妮,张璐.基于蚁群算法的测试用例集最小化研究.计算机工程,2009,35(6):213−215.

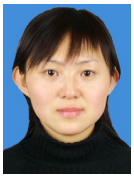
[6] 陈翔,陈继红,鞠小林,顾庆.回归测试中的测试用例优先排序技术述评.软件学报,2013,24(8):1695−1712. http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

边毅(1986－),男,宁夏银川人,博士,CCF 学生会员,主要研究领域为基于搜索的软件回归测试.

李征(1974－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为基于搜索的软件工程,程序源代码分析.

袁方(1990－),男,硕士,主要研究领域为演化计算,软件回归测试.

赵瑞莲(1964－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件可靠性分析.



郭俊霞(1977－),女,博士,讲师,CCF 高级会员,主要研究领域为网络信息定向抽取技术及测试.