

Referências Bibliográficas

- [Alameh2007] ALAMEH, R.; ZAZWORKA, N. ; HOLLINGSWORTH, J. K.. **Performance measurement of novice HPC programmers code**. In: SE-HPC '07: PROCEEDINGS OF THE 3RD INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR HIGH PERFORMANCE COMPUTING APPLICATIONS, p. 3, Washington, DC, USA, 2007. IEEE Computer Society. 5.2
- [Alind2008] ALIND, M.; ERIKSSON, M. V. ; KESSLER, C. W.. **Blocklib: a skeleton library for cell broadband engine**. In: IWMSE '08: PROCEEDINGS OF THE 1ST INTERNATIONAL WORKSHOP ON MULTICORE SOFTWARE ENGINEERING, p. 7–14, New York, NY, USA, 2008. ACM. 1, 5.3
- [Andrews2000] ANDREWS, G. R.. **Foundations of multithreaded, parallel, and distributed programming**. Addison-Wesley, 1 edition, 2000. 2.5, 4.3.2, 5.1
- [Arandi2011] ARANDI, S.; EVRIPIDOU, P.. **Ddm-vmc: the data-driven multithreading virtual machine for the cell processor**. In: PROCEEDINGS OF THE 6TH INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE AND EMBEDDED ARCHITECTURES AND COMPILERS, Hi-PEAC '11, p. 25–34, New York, NY, USA, 2011. ACM. 1.1, 5.2
- [Augonnet2011] AUGONNET, C.; THIBAUT, S.; NAMYST, R. ; WACRENIER, P.-A.. **Starpu: a unified platform for task scheduling on heterogeneous multicore architectures**. *Concurr. Comput. : Pract. Exper.*, 23:187–198, February 2011. 3.3, 5.2
- [Baduel2002] BADUEL, L.; BAUDE, F. ; CAROMEL, D.. **Efficient, flexible, and typed group communications in Java**. In: JGI '02: PROCEEDINGS OF THE 2002 JOINT ACM-ISCOPE CONFERENCE ON JAVA GRANDE, p. 28–36, New York, NY, USA, 2002. ACM. 5.2
- [Baduel2005] BADUEL, L.; BAUDE, F. ; CAROMEL, D.. **Object-oriented SPMD**. In: CCGRID '05: PROCEEDINGS OF THE FIFTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID

- (CCGRID'05) - VOLUME 2, p. 824–831, Washington, DC, USA, 2005. IEEE Computer Society. 5.2
- [Baker1999] BAKER, M.; CARPENTER, B.; FOX, G.; KO, S. H. ; LIM, S.. **mpi-java: an object-oriented java interface to mpi**. In: PROC. INTERNATIONAL WORKSHOP ON JAVA FOR PARALLEL AND DISTRIBUTED COMPUTING, IPPS/SPDP, San Juan, Puerto Rico, 1999. 2.2.2
- [Baude2007] BAUDE, F.; CAROMEL, D.; HENRIO, L. ; MOREL, M.. **Collective interfaces for distributed components**. In: CCGRID '07: PROCEEDINGS OF THE SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, p. 599–610, Washington, DC, USA, 2007. IEEE Computer Society. 5.2
- [Bentley2000] BENTLEY, J.. **Programming Pearls**. Addison-Wesley, 2 edition, 2000. 2.5.1
- [Bhansali2005] BHANSALI, P.. **Complexity measurement of data and control flow**. ACM SIGSOFT Software Engineering Notes, 3(1), 2005. 2.5.3
- [Bigot2007] BIGOT, J.; PEREZ, C.. **Enabling collective communications between components**. In: COMPFRAME '07: PROCEEDINGS OF THE 2007 SYMPOSIUM ON COMPONENT AND FRAMEWORK TECHNOLOGY IN HIGH-PERFORMANCE AND SCIENTIFIC COMPUTING, p. 121–130, New York, NY, USA, 2007. ACM. 5.2
- [Carriero1989] CARRIERO, N.; GELERNTER, D.. **How to write parallel programs: a guide to the perplexed**. ACM Comput. Surv., 21(3):323–357, 1989. 2.5.1
- [Carvalho2005] DE CARVALHO, F. H.; LINS, R. D.. **The sharp model: separation of concerns for reconciling modularity, abstraction and efficiency in distributed parallel programming**. In: SAC '05: PROCEEDINGS OF THE 2005 ACM SYMPOSIUM ON APPLIED COMPUTING, p. 1357–1364, New York, NY, USA, 2005. ACM. 1
- [Catanzaro2011] CATANZARO, B.; GARLAND, M. ; KEUTZER, K.. **Copperhead: compiling an embedded data parallel language**. In: PROCEEDINGS OF THE 16TH ACM SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, PPOPP '11, p. 47–56, New York, NY, USA, 2011. ACM. 2.5, 5.1, 5.2

- [Charles2005] 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: OOPSLA '05: PROCEEDINGS OF THE 20TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, p. 519–538, New York, NY, USA, 2005. ACM. 5.2
- [Chamberlain2007] CHAMBERLAIN, B. L.; CALLAHAN, D. ; ZIMA, H. P.. **Parallel programmability and the chapel language**. International Journal of High Performance Computing Applications, 21(3):291–312, 2007. Special Issue on High Productivity Programming Languages and Models. 1.1
- [Chafi2011] CHAFI, H.; SUJEETH, A. K.; BROWN, K. J.; LEE, H.; ATREYA, A. R. ; OLUKOTUN, K.. **A domain-specific approach to heterogeneous parallelism**. In: PROCEEDINGS OF THE 16TH ACM SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, PPOPP '11, p. 35–46, New York, NY, USA, 2011. ACM. 2.5, 5.2
- [Chakravarty2011] CHAKRAVARTY, M. M.; KELLER, G.; LEE, S.; MCDONELL, T. L. ; GROVER, V.. **Accelerating haskell array codes with multicore gpus**. In: PROCEEDINGS OF THE SIXTH WORKSHOP ON DECLARATIVE ASPECTS OF MULTICORE PROGRAMMING, DAMP '11, p. 3–14, New York, NY, USA, 2011. ACM. 5.1, 5.2
- [Danis2006] DANIS, C.. **Forms of collaboration in high performance computing: exploring implications for learning**. In: CSCW '06: PROCEEDINGS OF THE 2006 20TH ANNIVERSARY CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, p. 501–504, New York, NY, USA, 2006. ACM. 2.2.1
- [Dowell2008] DOSWELL, R.. **Python ps3**, 2008. 5.3
- [Evangelist1983] EVANGELIST, W. M.. **Relationships among computational, software, and intuitive complexity**. SIGPLAN Not., 18(12):57–59, 1983. 2.5.3
- [Faraj2005] FARAJ, A.; YUAN, X.. **Automatic generation and tuning of mpi collective communication routines**. In: ICS '05: PROCEEDINGS OF THE 19TH ANNUAL INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, p. 393–402, New York, NY, USA, 2005. ACM. 2.5.1, 2, 5.1

- [Faraj2006] FARAJ, A.; YUAN, X. ; LOWENTHAL, D. K.. **Star-mpi: self tuned adaptive routines for mpi collective operations**. In: ICS '06: PROCEEDINGS OF THE 20TH ANNUAL INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, p. 199–208, New York, NY, USA, 2006. ACM. 5.1
- [Fatahalian2006] FATAHALIAN, K.; HORN, D. R.; KNIGHT, T. J.; LEEM, L.; HOUSTON, M.; PARK, J. Y.; EREZ, M.; REN, M.; AIKEN, A.; DALLY, W. J. ; HANRAHAN, P.. **Sequoia: programming the memory hierarchy**. In: SC '06: PROCEEDINGS OF THE 2006 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, p. 83, New York, NY, USA, 2006. ACM. 5.3
- [Feng2011] FENG, M.; GUPTA, R. ; HU, Y.. **Spicec: scalable parallelism via implicit copying and explicit commit**. In: PROCEEDINGS OF THE 16TH ACM SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, PPOPP '11, p. 69–80, New York, NY, USA, 2011. ACM. 5.2
- [Ferrer2008] FERRER, R.; GONZÁLEZ, M.; SILLA, F.; MARTORELL, X. ; AY-GUADÉ, E.. **Evaluation of memory performance on the cell be with the sarc programming model**. In: PROCEEDINGS OF THE 9TH WORKSHOP ON MEMORY PERFORMANCE: DEALING WITH APPLICATIONS, SYSTEMS AND ARCHITECTURE, MEDEA '08, p. 77–84, New York, NY, USA, 2008. ACM. 5.3
- [Foster1995] FOSTER, I.. **Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 1, 2.5, 2.5.1, 5.1
- [Gelernter1992] GELERNTER, D.; CARRIERO, N.. **Coordination languages and their significance**. Commun. ACM, 35(2):97–107, 1992. 2.3
- [Gertz2003] GERTZ, E. M.; WRIGHT, S. J.. **Object-oriented software for quadratic programming**. ACM Trans. Math. Softw., 29(1):58–81, 2003. 2.5.2
- [Gorlatch2004] GORLATCH, S.. **Send-receive considered harmful: Myths and realities of message passing**. ACM Transactions on Programming Languages and Systems (TOPLAS), 26(1):47–56, 2004. 2.5.1, 5.1

- [Gropp1999] GROPP, W.; LUSKL, E. ; SKJELLUM, A.. **Using MPI: portable parallel programming with message passing interface**. MIT Press, 2 edition, 1999. 2.2.1, 2.2.2
- [Hinsen2006] HINSEN, K.; PETTER LANGTANGEN, H.; SKAVHAUG, O. ; SMUND ØDEGÅRD. **Using b sp and python to simplify parallel programming**. *Future Gener. Comput. Syst.*, 22:123–157, January 2006. 1, 2.3
- [IBM2005] IBM. **Bladecenter qs21**, 2005. 3.6
- [Ierusalimschy1996] IERUSALIMSCHY, R.; HENRIQUE, L.; WALDEMAR, F. ; FILHO, C.. **Lua - an extensible extension language**. *Software: Practice and Experience*, 26:635–652, 1996. 3.1
- [Jiao2003] JIAO, X.; CAMPBELL, M. T. ; HEATH, M. T.. **Rocom: an object-oriented, data-centric software integration framework for multiphysics simulations**. In: *ICS '03: PROCEEDINGS OF THE 17TH ANNUAL INTERNATIONAL CONFERENCE ON SUPERCOMPUTING*, p. 358–368, New York, NY, USA, 2003. ACM. 2.5.2
- [Josefsson2006] JOSEFSSON, S.. **The base16, base32, and base64 data encodings**, Oct. 2006. 3.8
- [Kahle2005] KAHLE, J. A.; DAY, M. N.; HOFSTEE, H. P.; JOHNS, C. R.; MAEURER, T. R. ; SHIPPY, D.. **Introduction to the cell multiprocessor**. *IBM Journal of Research and Development*, 49(4/5):589–604, 2005. 1.1, 3.6
- [Karwande2003] KARWANDE, A.; YUAN, X. ; LOWENTHAL, D. K.. **Cc-mpi: a compiled communication capable mpi prototype for ethernet switched clusters**. In: *PPOPP '03: PROCEEDINGS OF THE NINTH ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING*, p. 95–106, New York, NY, USA, 2003. ACM. 1, 2.5.1, 2, 5.1, 5.2
- [Ke2004] KE, J.; BURTSCHER, M. ; SPEIGHT, E.. **Runtime compression of mpi messanes to improve the performance and scalability of parallel applications**. In: *SC '04: PROCEEDINGS OF THE 2004 ACM/IEEE CONFERENCE ON SUPERCOMPUTING*, p. 59, Washington, DC, USA, 2004. IEEE Computer Society. 1, 2.5.1, 1, 5.1, 5.2
- [Kernighan1999] KERNIGHAN, B. W.; PIKE, R.. **The Practice of Programming**. Reading, MA, USA, 1999. 2.5.1

- [Khoury2011] KHOURY, R.; BURGSTALLER, B. ; SCHOLZ, B.. **Accelerating the execution of matrix languages on the cell broadband engine architecture.** IEEE Trans. Parallel Distrib. Syst., 22:7–21, January 2011. 1.1, 5.2
- [Khronos2010] KHRONOS. **The opencl specification**, 2010. 2.1
- [Kunzman2009] KUNZMAN, D. M.; KALÉ, L. V.. **Towards a framework for abstracting accelerators in parallel applications: experience with cell.** In: PROCEEDINGS OF THE CONFERENCE ON HIGH PERFORMANCE COMPUTING NETWORKING, STORAGE AND ANALYSIS, SC '09, p. 54:1–54:12, New York, NY, USA, 2009. ACM. 5.3
- [Kurzak2008] KURZAK, J.; BUTTARI, A.; LUSZCZEK, P. ; DONGARRA, J.. **The playstation 3 for high-performance scientific computing.** Computing in Science and Engg., 10(3):84–87, 2008. 3.6
- [Li2011] LI, J.; MA, X.; YOGINATH, S.; KORA, G. ; SAMATOVA, N. F.. **Transparent runtime parallelization of the r scripting language.** J. Parallel Distrib. Comput., 71:157–168, February 2011. 1, 2.3
- [Luszczek2007] LUSZCZEK, P.; DONGARRA, J.. **High performance development for high end computing with python language wrapper (plw).** Int. J. High Perform. Comput. Appl., 21:360–369, August 2007. 1, 2.3, 2.5
- [Mattson2004] MATTSON, T.; SANDERS, B. ; MASSINGILL, B.. **Patterns for parallel programming.** Addison-Wesley Professional, 2004. 2.4, 3.3, 5.2
- [Matthey2004] MATTHEY, T.; CICKOVSKI, T.; HAMPTON, S.; KO, A.; MA, Q.; NYERGES, M.; RAEDER, T.; SLABACH, T. ; IZAGUIRRE, J. A.. **Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics.** ACM Trans. Math. Softw., 30(3):237–265, 2004. 2.5.2, 5.2
- [McIlroy2010] MCILROY, R.; SVENTEK, J.. **Hera-jvm: a runtime system for heterogeneous multi-core architectures.** SIGPLAN Not., 45:205–222, October 2010. 1, 1.1, 2.3, 5.3
- [Miller2002] MILLER, P.. **Parallel, distributed scripting with python.** Oct. 2002. 1, 2.2.2, 2.3, 5.1
- [Mohamed2002] MOHAMED, N.; AL-JAROODI, J.; JIANG, H. ; SWANSON, D.. **Jopi: a java object-passing interface.** In: JGI '02: PROCEEDINGS

- OF THE 2002 JOINT ACM-ISCOPE CONFERENCE ON JAVA GRANDE, p. 37–45, New York, NY, USA, 2002. ACM. 2.2.2, 5.2
- [Mueller2007] MUELLER, C.. **Synthetic Programming: User-Directed Run-Time Code Synthesis for High Performance Computing**. PhD thesis, Indiana University, 2007. 3.6, 5.3
- [Nehab2007] NEHAB, D.. **Luasocket**, 2007. 3.8, 5.2
- [Norton1995] NORTON, C. D.; SZYMANSKI, B. K. ; DECYK, V. K.. **Object-oriented parallel computation for plasma simulation**. Commun. ACM, 38(10):88–100, 1995. 2.5.2
- [Ong2002] ONG, E.. **Mpi ruby: Scripting in a parallel environment**. Computing in Science and Engg., 4(4):78–82, 2002. 1, 2.2.2, 2.3
- [Pall2011] PALL, M.. **Luajit**, 2011. 3.7, 4.5
- [Parri2011] PARRI, J.; SHAPIRO, D.; BOLIC, M. ; GROZA, V.. **Returning control to the programmer: Simd intrinsics for virtual machines**. Queue, 9:30:30–30:37, February 2011. 2.3
- [Perez2007] PEREZ, J. P.; BELLENS, P.; BADIA, R. M. ; LABARTA, J.. **Cellss: making it easier to program the cell broadband engine processor**. IBM J. Res. Dev., 51(5):593–604, 2007. 5.3
- [Peters1998] PETERS, J. F.; PEDRYCZ, W.. **Software Engineering: An Engineering Approach**. John Wiley & Sons, Inc., New York, NY, USA, 1998. 1
- [Pressman1997] PRESSMAN, R. S.. **Software Engineering: A Practitioner's Approach**. McGraw-Hill Higher Education, 1997. 1, 2.5.3
- [Rabbah2007] RABBAH, R.. **Beyond gaming: programming the playstation®3 cell architecture for cost-effective parallel processing**. In: CODES+ISSS '07: PROCEEDINGS OF THE 5TH IEEE/ACM INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODE-SIGN AND SYSTEM SYNTHESIS, p. 1–1, New York, NY, USA, 2007. ACM. 3.6
- [Rosling1999] ROSING, M.; YABUSAKI, S.. **A programmable preprocessor for parallelizing fortran-90**. In: PROCEEDINGS OF THE 1999 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, New York, NY, USA, 1999. ACM. 5.1

- [Sankaralingam2010] SANKARALINGAM, K.; ARPACI-DUSSEAU, R. H.. **Get the parallelism out of my cloud**. In: PROCEEDINGS OF THE 2ND USENIX CONFERENCE ON HOT TOPICS IN PARALLELISM, HotPar'10, p. 8–8, Berkeley, CA, USA, 2010. USENIX Association. 2.3, 2.5, 5.1
- [Skillicorn1998] SKILLICORN, D. B.; TALIA, D.. **Models and languages for parallel computation**. ACM Comput. Surv., 30(2):123–169, 1998. 1, 3, 5.1
- [Sommerville2001] SOMMERVILLE, I.. **Software engineering (6th ed.)**. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2001. 1, 2.2.2, 2.5.2
- [Sony2005] SONY. **Playstation 3**, 2005. 3.6
- [Stamatakis2008] STAMATAKIS, A.; OTT, M.. **Exploiting fine-grained parallelism in the phylogenetic likelihood function with mpi, pthreads, and openmp: A performance study**. In: PROCEEDINGS OF THE THIRD IAPR INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION IN BIOINFORMATICS, PRIB '08, p. 424–435, Berlin, Heidelberg, 2008. Springer-Verlag. 3.7
- [Szyperski2003] SZYPERSKI, C.. **Component technology: what, where, and how?** In: ICSE '03: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 684–693, Washington, DC, USA, 2003. IEEE Computer Society. 2.5.2
- [Tan2003] TAN, K.; SZAFRON, D.; SCHAEFFER, J.; ANVIK, J. ; MACDONALD, S.. **Using generative design patterns to generate parallel code for a distributed memory environment**. In: PPOPP '03: PROCEEDINGS OF THE NINTH ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, p. 203–215, New York, NY, USA, 2003. ACM. 2.5.1, 2, 5.1
- [Tsoi2011] TSOI, K. H.; TSE, A. H.; PIETZUCH, P. ; LUK, W.. **Programming framework for clusters with heterogeneous accelerators**. SIGARCH Comput. Archit. News, 38:53–59, January 2011. 2.5, 3.3, 5.2
- [Ururahy1999] URURAHY, C.; RODRIGUEZ, N.. **Alua: An event-driven communication mechanism for parallel and distributed programming**. In: PROCEEDINGS OF ISCA 12TH INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS (PDCS-99), p. 108–113, New York, NY, USA, 1999. 1, 2.3

- [Vadhiyar2000] VADHIYAR, S. S.; FAGG, G. E. ; DONGARRA, J.. **Automatically tuned collective communications**. In: SUPERCOMPUTING '00: PROCEEDINGS OF THE 2000 ACM/IEEE CONFERENCE ON SUPERCOMPUTING (CDROM), p. 3, Washington, DC, USA, 2000. IEEE Computer Society. 5.1
- [Vishkin2011] VISHKIN, U.. **Using simple abstraction to reinvent computing for parallelism**. Commun. ACM, 54:75–85, January 2011. 2.5, 5.1, 5.4
- [Williams2008] WILLIAMS, K.; NOLL, A.; GAL, A. ; GREGG, D.. **Optimization strategies for a java virtual machine interpreter on the cell broadband engine**. In: PROCEEDINGS OF THE 5TH CONFERENCE ON COMPUTING FRONTIERS, CF '08, p. 189–198, New York, NY, USA, 2008. ACM. 1, 2.3, 5.3
- [Zhong2007] ZHONG, W.; ALTUN, G.; TIAN, X.; HARRISON, R.; TAI, P. C. ; PAN, Y.. **Parallel protein secondary structure prediction schemes using pthread and openmp over hyper-threading technology**. J. Supercomput., 41:1–16, July 2007. 3.7
- [nVidia2007] NVIDIA NVIDIA, D. Z.. **Cuda**, 2007. 2.1

A

API Numina em Detalhes

Apresentamos aqui a descrição completa da API Numina. O desenvolvedor pode utilizar diretamente o nível mais baixo que expõe as chamadas nativas para Lua, no entanto recomenda-se o uso das abstrações de mais alto nível. A seguir apresentamos primeiro as chamadas de mais baixo nível e em seguida as chamadas que podem ser feitas em cada abstração.

A.1

API de Sistema

1. `new(id)` - responsável por criar o ambiente numina inicial, já inicia as máquinas virtuais Lua em cada thread do sistema. As threads têm uma correspondência de um para um com os núcleos do sistema e podem ser indexadas por um índice numérico de 1 até N.
2. `loaddump(ref, função, nome, id)` - função responsável por carregar o código executável de uma função do usuário em uma thread do sistema. Para isso deve receber uma referência ao ambiente numina, o código da função em formato string, o nome da função e o identificador do núcleo.
3. `open(id)` - inicia uma máquina virtual Lua no núcleo indicado, essa função já é chamada pela função `new`, mas pode ser utilizada caso o usuário precise reiniciar o sistema sem precisar reiniciar as threads de processamento.
4. `execute(nome, id, ...)` - permite executar a função no núcleo indicado passando os parâmetros recebidos.
5. `checkstate(id)` - verifica se a execução da função já encerrou no núcleo indicado. Esta função atua de forma bloqueante.
6. `getsingleresult(id)` - permite recuperar o resultado do processamento de uma função no núcleo indicado.
7. `corecount()` - retorna a quantidade de núcleos nesse ambiente numina, essa função é utilizada pela abstração `CoreGroup`.

8. `close(id)` - permite encerrar a máquina virtual Lua no núcleo indicado sem encerrar a thread de processamento.
9. `exit()` - encerra o ambiente numina terminando as threads de execução.
10. `loadnativearray(ref, nativearray, nome, id)` - permite carregar um native array no núcleo indicado.
11. `getnativearray(ref, nome, id)` - permite recuperar um native array do núcleo indicado.
12. `destroynativearray(ref, nome, id)` - permite remover um native array do núcleo indicado.

A.2 API Core

Com a primeira abstração utilizamos basicamente a mesma API do Sistema, porém com algumas referências e índices já atribuídos, isto porque quando criamos um objeto Core já indicamos qual núcleo ele representará. Tipicamente essa abstração é referenciada indiretamente quando o programador usa a abstração CoreGroup.

1. `Core:new(id)` - cria um Core com o identificador especificado e passa a controlar a thread de mesmo identificador no ambiente numina.
2. `Core:load(func,nome)` - permite carregar uma função na máquina virtual Lua desse núcleo e incorpora a função à lista de operações desse núcleo de forma que a função possa ser chamada pelo nome diretamente.
3. `Core:exec(nome,...)` - essa função é utilizada pela abstração CoreGroup para executar uma função que tenha sido definida para esse Core.
4. `Core:loadnativearray(array,name)` - permite carregar um nativearray neste Core.
5. `Core:getnativearray(array, name)` - permite recuperar um nativearray deste Core.
6. `Core:destroynativearray(name)` - permite remover um nativearray deste Core.
7. `Core:checkstate()` - permite verificar se a chamada de uma função já encerrou.

8. `Core:getsingleresult()` - permite recuperar o valor de processamento de uma função neste Core.
9. `Core:open()` - permite iniciar a máquina virtual Lua deste Core.
10. `Core:close()` - permite encerrar a máquina virtual Lua deste Core sem terminar a thread de processamento.
11. `Core:count()` - retorna a quantidade de núcleos neste ambiente numina, utilizada pela abstração `CoreGroup`.
12. `Core:exit()` - permite encerrar as threads de processamento do ambiente numina.

A.3

API `CoreGroup`

A abstração `CoreGroup` agrupa um conjunto de núcleos que estejam em uma mesma máquina, dessa forma permite a criação de algumas novas funcionalidades para agregação de dados.

1. `CoreGroup:new(id)` - cria um `CoreGroup` com o identificador especificado e cria os objetos `Core` para corresponder às threads de mesmo identificador no ambiente numina.
2. `CoreGroup:add(core)` - permite adicionar um objeto `Core` a este `CoreGroup`.
3. `CoreGroup:getCore(id)` - retorna uma referência ao objeto `Core` de índice `id` neste `CoreGroup`.
4. `CoreGroup:load(func,name,...)` - permite carregar uma função em todos os objetos `Core` deste `CoreGroup` e incorpora o nome da função ao objeto `CoreGroup` para que o usuário possa fazer uma chamada direta.
5. `CoreGroup:exec(n,args)` - esta função é usada pela abstração `Cluster` para executar uma função que foi definida neste `CoreGroup`.
6. `CoreGroup:checkstate()` - permite verificar se a chamada de uma função já encerrou. Neste caso cada um dos objetos `Core` serão verificados.
7. `CoreGroup:gather()` - permite retornar uma tabela Lua contendo os valores de processamento de cada `Core`.

8. `CoreGroup:reduce(func)` - permite recuperar os valores de retorno de cada Core, mas aplicando uma função para tratamento dos resultados de acordo com os itens abaixo:
 - SUM - soma todos os resultados
 - PROD - executa um produto de todos os resultados
 - MAX - retorna o maior de todos os resultados
 - MIN - retorna o menor de todos os resultados
9. `CoreGroup:close()` - permite encerrar todas as máquinas virtuais Lua dos objetos Core deste CoreGroup sem encerrar as threads de processamento.
10. `CoreGroup:getCount()` - retorna a quantidade de objetos Core neste CoreGroup.
11. `CoreGroup:exit()` - encerra as threads de processamento do ambiente numina.

A.4 API Cluster

A abstração Cluster foi criada para permitir o uso de vários CoreGroups de forma coordenada, para isso ela se comunica com um daemon que fica em cada máquina participante. A API é muito semelhante a do CoreGroup e por isso vamos listar apenas as novas funções.

1. `Cluster:getNodeCount()` - retorna a quantidade de CoreGroups neste Cluster.
2. `Cluster:getTotalCoreCount()` - retorna a quantidade total de Cores neste Cluster.
3. `Cluster:getNodeCoreCount(i)` - retorna a quantidade de Cores no Core-Group indicado.

B

Código Fonte das Aplicações para Análise Qualitativa

Apresentamos aqui o código completo das versões C, Java e Numina da aplicação de aproximação de pi com método de Monte-Carlo. Por questões de ajustes de página algumas linhas de código C e Java foram quebradas em mais de uma linha, mas foram consideradas como uma única linha para efeito de comparação.

Listagem B.1: Código C

```
1 #include <pthread.h>
2 #include <sched.h>
3 #include <sys/resource.h>
4 #include <sys/time.h>
5 #include <math.h>
6 #include <stdlib.h>
7 #include <stdio.h>
8 #define MAXTHREADS 4
9 pthread_mutex_t imutex[MAXTHREADS];
10 pthread_cond_t icondition[MAXTHREADS];
11 int ibox[MAXTHREADS];
12 #define ARRAY_SIZE 4000
13 #define MAX 20000000
14 typedef struct pi_data {
15     int core;
16     double somaparcial;
17     double data[ARRAY_SIZE];
18 } pi_data_t;
19 pi_data_t dados[MAXTHREADS];
20 double getnumber() {
21     int n = random();
22     double f = ((double) n) / RAND_MAX;
23     return f;
24 }
25 void calcp(pi_data_t *d) {
```

```
26     int core = d->core;
27     int tam = ARRAY_SIZE / 2;
28     while (1) {
29         pthread_mutex_lock(&imutex[core]);
30         while (ibox[core] != core) {
31             pthread_cond_wait(&icondition[core], &imutex[core]);
32         }
33         core = d->core;
34         if (core == -1) break;
35         double soma = d->somaparcial;
36         double *numeros = d->data;
37         int j = 0;
38         for (j = 0; j < tam; j++) {
39             double randomX = numeros[j];
40             double randomY = numeros[j + 2000];
41             if ((randomY * randomY)+(randomX * randomX) <= 1)
42                 soma++;
43         }
44         d->somaparcial = soma;
45         ibox[core] = -1;
46         pthread_cond_broadcast(&icondition[core]);
47         pthread_mutex_unlock(&imutex[core]);
48     }
49 }
50 int main(int argc, char *argv[]) {
51     int REPS_PER_SPU = (MAX / MAXTHREADS) / (ARRAY_SIZE / 2);
52     int N = 0;
53     int i = 0;
54     int j = 0;
55     int q = 0;
56     N = 20000000;
57     for (i = 0; i < MAXTHREADS; i++) {
58         pthread_mutex_init(&(imutex[i]), NULL);
59         pthread_cond_init(&(icondition[i]), NULL);
60     }
61     int status = -1;
62     pthread_t pthread[MAXTHREADS];
63     for (j = 0; j < MAXTHREADS; j++) {
64         for (i = 0; i < ARRAY_SIZE; i++) {
```

```
65         dados[j].data[i] = getnumber();
66     }
67 }
68 for (i = 0; i < MAXTHREADS; i++) {
69     dados[i].core = i;
70     ibox[i] = i;
71     status = pthread_create(
72         &pthread[i], NULL, &calcpi, &dados[i]);
73 }
74 for (i = 0; i < REPS_PER_SPU; i++) {
75     for (j = 0; j < MAXTHREADS; j++) {
76         pthread_mutex_lock(&imutex[j]);
77         while (ibox[j] != -1) {
78             pthread_cond_wait(&icondition[j], &imutex[j]);
79         }
80         for (q = 0; q < ARRAY_SIZE; q++) {
81             dados[j].data[q] = getnumber();
82         }
83         ibox[j] = j;
84         pthread_cond_broadcast(&icondition[j]);
85         pthread_mutex_unlock(&imutex[j]);
86     }
87 }
88 for (j = 0; j < MAXTHREADS; j++) {
89     pthread_mutex_lock(&imutex[j]);
90     while (ibox[j] != -1) {
91         pthread_cond_wait(&icondition[j], &imutex[j]);
92     }
93     dados[j].core = -1;
94     ibox[j] = j;
95     pthread_cond_broadcast(&icondition[j]);
96     pthread_mutex_unlock(&imutex[j]);
97 }
98 for (i = 0; i < MAXTHREADS; i++) {
99     void *ret;
100     pthread_join(pthread[i], &ret);
101 }
102 double rsoma = dados[0].somaparcial +
103     dados[1].somaparcial;
```



```

104     rsoma = (rsoma / N)*4;
105     for (i = 0; i < MAXTHREADS; i++) {
106         pthread_mutex_destroy(&(imutex[i]));
107         pthread_cond_destroy(&(icondition[i]));
108     }
109     return 0;
110 }

```

Listagem B.2: Código Java

```

1 import java.util.logging.Level;
2 import java.util.logging.Logger;
3 public class Multiply {
4     public static void main(String[] args) {
5         int MAX_THREADS = 4;
6         int ARRAY_SIZE = 4000;
7         int N = 20000000;
8         int REPS_PER_SPU = (N / MAX_THREADS) / (ARRAY_SIZE / 2);
9         double numeros[][] = new double[MAX_THREADS][ARRAY_SIZE];
10        Thread cores[] = new Thread[MAX_THREADS];
11        Calculador calculadores[] = new Calculador[MAX_THREADS];
12        Integer monitores[][] = new Integer[MAX_THREADS][1];
13        for (int i = 0; i < cores.length; i++) {
14            for (int j = 0; j < numeros.length; j++) {
15                numeros[i][j] = Math.random();
16            }
17        }
18        for (int i = 0; i < cores.length; i++) {
19            monitores[i][0] = i;
20            calculadores[i] =
21                new Calculador(i, monitores[i], numeros[i]);
22            cores[i] = new Thread(calculadores[i]);
23            cores[i].start();
24        }
25        for (int i = 0; i < REPS_PER_SPU; i++) {
26            for (int j = 0; j < MAX_THREADS; j++) {
27                synchronized (monitores[j]) {
28                    while (monitores[j][0] != -1) {
29                        try {
30                            monitores[j].wait();

```

```
31         } catch (InterruptedException ex) {
32             Logger.getLogger(
33                 Multiply.class.getName()).log(
34                     Level.SEVERE, null, ex);
35         }
36     }
37     for (int q = 0; q < numeros.length; q++) {
38         numeros[j][q] = Math.random();
39     }
40     monitores[j][0] = j;
41     monitores[j].notifyAll();
42 }
43 }
44 }
45 double soma = calculadores[0].getSoma() +
46             calculadores[1].getSoma();
47 for (int j = 0; j < MAX_THREADS; j++) {
48     synchronized (monitores[j]) {
49         while (monitores[j][0] != -1) {
50             try {
51                 monitores[j].wait();
52             } catch (InterruptedException ex) {
53                 Logger.getLogger(
54                     Multiply.class.getName()).log(
55                         Level.SEVERE, null, ex);
56             }
57         }
58         calculadores[j].setTerminar(true);
59         monitores[j].notifyAll();
60     }
61     try {
62         cores[j].join();
63     } catch (InterruptedException ex) {
64         Logger.getLogger(
65             Multiply.class.getName()).log(
66                 Level.SEVERE, null, ex);
67     }
68 }
69 soma = (soma / N) * 4;
```

```
70     }
71 }
72
73 import java.util.logging.Level;
74 import java.util.logging.Logger;
75 public class Calculador implements Runnable {
76     private int core;
77     private double num[];
78     private final Integer calc[];
79     private double soma;
80     private int tam;
81     private boolean terminar;
82     Calculador(int core, Integer object[], double num[]) {
83         this.core = core;
84         this.calc = object;
85         this.num = num;
86         tam = num.length / 2;
87     }
88     public Object getCalc() {
89         return calc;
90     }
91     public void run() {
92         while (!isTerminar()) {
93             synchronized (calc) {
94                 while (calc[0] != core) {
95                     try {
96                         if (terminar) break;
97                         calc.wait();
98                     } catch (InterruptedException ex) {
99                         Logger.getLogger(
100                             Calculador.class.getName()).log(
101                                 Level.SEVERE, null, ex);
102                     }
103                 }
104                 for (int i = 0; i < tam; i++) {
105                     double randomX = num[i];
106                     double randomY = num[i + 2000];
107                     if ((randomY * randomY) +
108                         (randomX * randomX) <= 1) {
```

```

109             soma++;
110         }
111     }
112     calc[0] = -1;
113     calc.notifyAll();
114 }
115 }
116 }
117 public double getSoma() {
118     return soma;
119 }
120 public boolean isTerminar() {
121     return terminar;
122 }
123 public void setTerminar(boolean terminar) {
124     this.terminar = terminar;
125 }
126 }

```

Listagem B.3: Código Numina

```

1 require "nativearray"
2 require "coregroup"
3 group = CoreGroup:new(1)
4 SPUS = group:getCount()
5 ARRAY_SIZE = 4000
6 MAX = 20000000
7 REPS_PER_SPU = (MAX / SPUS) / (ARRAY_SIZE/2)
8 float = 1
9 local tableX = {}
10 for i=1,SPUS do
11     tableX[i] = nativearray.new(ARRAY_SIZE,float)
12 end
13 function getnumber()
14     return math.random();
15 end
16 function calcula(size, s)
17     require "nativearray"
18     local i = 0;
19     local soma = s;

```

```
20         for i=1,size do
21             local x = nativearray.get(rX,i)
22             local y = nativearray.get(rX,(i + 2000))
23             if ((y * y)+(x * x) <= 1) then
24                 soma = soma + 1;
25             end
26         end
27         return soma;
28 end
29 function main()
30     local N = 0;
31     local i = 0;
32     local j = 0;
33     N = 20000000;
34     local soma = {0,0,0,0};
35     for i = 1,SPUS do
36         for j = 1,ARRAY_SIZE do
37             nativearray.set(tableX[i],j,getnumber())
38         end
39     end
40     for j = 1,SPUS do
41         group:getCore(j):loadnativearray(tableX[j], "rX")
42     end
43     for i = 1,REPS_PER_SPU do
44         group:calcula(ARRAY_SIZE/2, soma)
45         soma = group:gather()
46         for i = 1,SPUS do
47             for j = 1,ARRAY_SIZE do
48                 nativearray.set(tableX[i],j,getnumber())
49             end
50         end
51     end
52     rsoma = soma[1] +soma[2] +soma[3] +soma[4]
53     rsoma = (rsoma / N)*4;
54 end
55 group:load(calcula, "calcula", "broadcast", "scatter")
56 main()
57 group:close()
58 group:exit()
```

C

Aplicações para Comparação de Tempos entre Camadas

Listagem C.1: Uso da API Numina Diretamente

```
1 require "numina"
2
3 SPUS = 4
4 numen = numina.new()
5 for i=1,SPUS do
6     numina.open(numen,i);
7 end
8
9 function primo(id, ini, fim)
10     for i = ini,fim,2 do
11         if(testaprimo(i)) then
12             end
13     end
14 end
15
16 function testaprimo(p1)
17     a = 3;
18     limite = math.floor(math.sqrt(p1));
19     for a=3,limite,2 do
20         if ((p1 - math.floor(p1/a) * a) == 0) then
21             return false;
22         end
23     end
24     return true;
25 end
26
27 function main()
28     for k = 1,SPUS do
29         numina.loaddump(numen, testaprimo, "testaprimo" ,k)
30         numina.loaddump(numen, primo, "primo" ,k)
```

```

31     end
32
33     ini = 999999999999901
34     if arg[1] ~= nil then
35         ini = arg[1]
36     end
37     fim = 999999999999999
38     tmp = (fim - ini) / SPUS;
39     tmpFim = ini + tmp;
40     numina.execute(numen,1,"primo",1,ini,tmpFim)
41
42     ini = math.ceil(tmpFim+1)
43     tmpFim = ini + tmp
44     numina.execute(numen,2,"primo",2,ini,tmpFim)
45
46     ini = math.ceil(tmpFim+1)
47     tmpFim = ini + tmp
48     numina.execute(numen,3,"primo",3,ini,tmpFim)
49
50     ini = math.ceil(tmpFim+1)
51     tmpFim = ini + tmp
52     numina.execute(numen,4,"primo",4,ini,tmpFim)
53
54     for k = 1,SPUS do
55         numina.checkstate(numen,k)
56     end
57 end
58
59 main()
60
61 for i=1,SPUS do
62     numina.close(numen,i)
63 end
64 numina.exit(numen)

```

Listagem C.2: Uso da Abstração Core

```

1 require "core"
2
3 SPUS = 4

```

```
4 cores = {}
5 for i=1,SPUS do
6     cores[i] = Core:new(i)
7 end
8
9 function primo(id, ini, fim)
10     for i = ini,fim,2 do
11         if(testaprimo(i)) then
12             end
13         end
14 end
15
16 function testaprimo(p1)
17     a = 3;
18     limite = math.floor(math.sqrt(p1));
19     for a=3,limite,2 do
20         if ((p1 - math.floor(p1/a) * a) == 0) then
21             return false;
22         end
23     end
24     return true;
25 end
26
27 function main()
28     for k = 1,SPUS do
29         cores[k]:load(testaprimo, "testaprimo")
30         cores[k]:load(primo, "primo")
31     end
32
33     ini = 99999999999901
34     if arg[1] ~= nil then
35         ini = arg[1]
36     end
37     fim = 99999999999999
38     tmp = (fim - ini) / SPUS;
39     tmpFim = ini + tmp;
40     cores[1]:primo(1,ini,tmpFim)
41
42     ini = math.ceil(tmpFim+1)
```



```

43     tmpFim = ini + tmp
44     cores[2]:primo(2,ini,tmpFim)
45
46     ini = math.ceil(tmpFim+1)
47     tmpFim = ini + tmp
48     cores[3]:primo(3,ini,tmpFim)
49
50     ini = math.ceil(tmpFim+1)
51     tmpFim = ini + tmp
52     cores[4]:primo(4,ini,tmpFim)
53
54     for k = 1,SPUS do
55         cores[k]:checkstate()
56     end
57 end
58
59 main()
60
61 for i=1,SPUS do
62     cores[i]:close()
63 end
64 cores[1]:exit()

```

Listagem C.3: Uso da Abstração CoreGroup

```

1 require "coregroup"
2
3 group = CoreGroup:new(1)
4 SPUS = group:getCount()
5
6 function primo(id, ini, fim)
7     for i = ini,fim,2 do
8         if(testaprimo(i)) then
9             end
10        end
11 end
12
13 function testaprimo(p1)
14     a = 3;
15     limite = math.floor(math.sqrt(p1));

```

```
16         for a=3,limite,2 do
17             if ((p1 - math.floor(p1/a) * a) == 0) then
18                 return false;
19             end
20         end
21         return true;
22 end
23
24 function main()
25     group:load(testaprimo, "testaprimo","broadcast")
26     group:load(primo, "primo","scatter","scatter","scatter")
27
28     ini = 999999999999901
29     if arg[1] ~= nil then
30         ini = arg[1]
31     end
32     fim = 999999999999999
33     size = fim - ini
34     qtdPerCore = math.ceil((size/SPUS))
35     li = ini
36     lf = ini ---+ qtdPerCore[1]
37
38
39     parmsid = {}
40     parmsi = {}
41     parmsf = {}
42     id = 1;
43     for i=1,SPUS do
44         lf = li + qtdPerCore
45         parmsid[i] = id
46         parmsi[i] = li
47         parmsf[i] = lf
48         id = id + 1
49         li = lf + 1
50         if ((li - math.floor(li/2) * 2) == 0) then
51             li = li + 1
52         end
53     end
54 end
```

```
55         group:primo(parmsid,parmsi,parmsf)
56         group:checkstate()
57 end
58
59 main()
60
61 group:close()
62
63 group:exit()
```