Computação sobre dados cifrados em *GPGPUs*

Pedro Geraldo M. R. Alves, Diego F. Aranha

Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Cidade Universitária Zeferino Vaz – CEP: 13083-852 – Campinas – SP – Brazil

{pedro.alves,dfaranha}@ic.unicamp.br

Abstract. *Under the dominant cloud computing paradigm, employing encryption for data storage and transport may not be enough for assurance of secrecy, because the data owner has no real control over the processing hardware. This way, security guarantees should also be extended to data processing. Homomorphic encryption schemes are natural candidates for computation over encrypted data in the cloud, since they are able to satisfy all security requirements imposed by that environment. This work investigates strategies to efficiently implement the leveled fully homomorphic scheme* YASHE *on GPGPUs. As result of this research, the* CUYASHE *library was developed and made available to the community. In particular, it presents a speedup between* 6 *and* 35 *times for homomorphic multiplication. This operation is performance-critical for evaluating any function over encrypted data, demonstrating that GPGPUs are an appropriate technology for bootstrapping privacy-preserving cloud computing environments.*

Resumo. *No contexto da computação na nuvem, a aplicação de métodos criptográficos exclusivamente no armazenamento e transporte dos dados não é suficiente para a preservação de sigilo uma vez que os detentores dos dados não tem controle real sobre o hardware de processamento. Dessa forma, os requisitos de segurança precisam também ser estendidos para o processamento dos dados. Esquemas de cifração homomórfica são candidatos naturais para computação sobre dados cifrados na nuvem, visto que são capazes de satisfazer todos os requerimentos de segurança impostos pelo ambiente. Este trabalho investiga estratégias para implementação eficiente do criptossistema homomórfico em nível* YASHE *em GPGPUs. Como fruto das conclusões obtidas, a biblioteca* CUYASHE *foi desenvolvida e disponibilizada à comunidade. Em particular, ela apresenta ganhos de velocidade entre* 6 *e* 35 *vezes para a operação de multiplicação homomórfica. Por ser uma operação com desempenho critico para a avaliação de funções sobre criptogramas, esse resultado demonstra que GPGPUs são uma tecnologia apropriada para computação que preserve privacidade em ambientes de nuvem.*

1. Introdução

A computação em nuvem tem sido responsável por uma profunda mudança nas soluções de processamento distribuído. A possibilidade de terceirizar a instalação, manutenção e a escalabilidade de servidores, somada a preços competitivos, faz com que esses serviços se tornem altamente atraentes, tendo sua aplicação feita em setores diversos, desde o meio científico até o mercado de dispositivos móveis [Hoffa et al. 2008, Dinh et al. 2013].

A adoção da computação em nuvem, entretanto, levanta importantes questões de segurança. A possibilidade do vazamento de informações acompanha o crescimento do

número de entidades que manipulam os dados, e o sigilo é apontado como uma das maiores preocupações [Xiao and Xiao 2013].

Diversos esquemas criptográficos são utilizados como padrão para o armazenamento e transferência de dados. Contudo, no caso de serviços na nuvem existe a possibilidade de se lidar com um adversário que não apenas tenha acesso aos dados como também ao *hardware* que realiza seu processamento. Desse modo, é necessário que se implemente estratégias de proteção também durante o processamento.

A criptografia homomórfica se mostra promissora para satisfazer esse requisito de segurança. Os esquemas dessa classe permitem que o processamento seja feito sobre criptogramas mesmo sem o conhecimento das chaves de cifração ou decifração. Assim, não há razão para que dados sejam revelados no momento do processamento.

Objetivo. O objetivo deste trabalho consistiu em realizar uma implementação do criptossistema homomórfico YASHE [Bos et al. 2013] com ganho de velocidade nas operações homomórficas em relação ao estado da arte. A abordagem definida foi aplicar técnicas de paralelismo por meio de *GPGPUs* sobre a plataforma CUDA.

Contribuições. Este trabalho apresenta a primeira implementação em *GPGPUs* do criptossistema homomórfico em nível YASHE. Técnicas de implementação paralela em CUDA foram aplicadas no desenvolvimento da aritmética necessária e explorou-se propriedades especiais de sua estrutura algébrica com o intuito de reduzir a complexidade computacional de certas operações, como por exemplo redução polinomial e modular. Aplicou-se o Teorema Chinês do Resto (*CRT*) com o intuito de evitar o uso de aritmética multi-precisão e foi realizada análise de como a Transformada Rápida de Fourier, *FFT*, e a *Number-Theoretic Transform*, *NTT*, podem ser aplicadas na redução da complexidade de multiplicações polinomiais. Por fim, a biblioteca CUYASHE foi desenvolvida e disponibilizada à comunidade [Alves and Aranha 2016c]. Ela consolida as conclusões apresentadas neste documento e, em particular, demonstra ganhos de velocidade entre 6 até 35 vezes na multiplicação homomórfica em relação ao estado da arte, uma operação vital para o criptossistema.

Resultados preliminares deste trabalho foram apresentados no X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica em Andamento do Instituto de Computação da Unicamp (X WTD) [Alves and Aranha 2015a]; e no XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2015) [Alves and Aranha 2015b]. Suas conclusões foram apresentadas na defesa de mestrado do autor [Alves and Aranha 2016a] e no I Encontro de Teoria da Computação (ETC 2016) dentro do XXXVI Congresso da Sociedade Brasileira de Computação (CSBC 2016) [Alves and Aranha 2016b].

2. Fundamentação teórica

A implementação eficiente da aritmética de um criptossistema é fator fundamental para o ganho de velocidade em suas operações. Neste trabalho, foi necessário o estudo de estratégias para a implementação eficiente da aritmética polinomial sobre inteiros multi-precisão em *GPUs*. Essas estratégias envolveram a simplificação da implementação da aritmética por meio do *CRT* e o uso de variantes da Transformada Discreta de Fourier na redução da complexidade computacional de multiplicações polinomiais.

Teorema Chinês do Resto (*CRT*). O *CRT* foi utilizado como ferramenta para simplificar a manipulação dos coeficientes polinomiais. Por meio do teorema, um polinômio com coefi-

cientes inteiros arbitrariamente grandes é decomposto em um conjunto de polinômios com coeficientes inteiros arbitrariamente pequenos chamados polinômios residuais, ou simplesmente resíduos. Essa substituição permite que os coeficientes sejam manipulados utilizando uma aritmética mais simples e suportada nativamente pela arquitetura.

A aplicação desse teorema implica no encarecimento das operações aritméticas, uma vez que precisam ser replicadas entre os resíduos de cada um dos operandos. É esperado que esse problema seja absorvido pela capacidade de processamento paralelo da *GPU*.

Multiplicação polinômial. A quantidade de operações necessárias em um algoritmo simples para o cálculo de uma multiplicação polinomial tem custo $\Theta(N^2)$, para operandos de grau N, o que compromete a escalabilidade dessa operação. No contexto dos criptossistemas baseados no problema *RLWE*, como observado por Lindner e Peikert, a segurança é fortemente relacionada com o grau do anel de polinômios [Lindner and Peikert 2011]. Especificamente para o YASHE, Lepoint e Naehrig utilizam parâmetros com N variando de 2^{11} até 2^{16} para atingir um padrão de segurança equivalente a $\lambda = 80$ *bits* [Lepoint and Naehrig 2014]. Dessa forma, a multiplicação de polinômios de grau alto é uma operação vital para esses esquemas, o que implica que melhorias de velocidade geram impacto considerável na velocidade de suas operações.

Nesse sentido, duas transformadas, variantes da Transformada Discreta de Fourier e capazes de prover o ganho de velocidade almejado se destacam e foram estudas neste trabalho:

Transformada Rápida de Fourier (*FFT*). O algoritmo da Transformada Rápida de Fourier, ou *FFT*, pode ser empregado na redução da complexidade computacional de uma multiplicação polinomial [Cooley and Tukey 1965]. Utiliza aritmética do corpo dos números complexos e provê um domínio em que multiplicações polinomiais são reduzidas a multiplicação elemento-a-elemento. Dessa forma, considerando o custo da aplicação da transformada sobre os operandos, essa operação assume complexidade $\Theta(N \log N)$, o que representa uma melhoria significativa de velocidade.

Number-Theoretic Transform (*NTT*). Enquanto a *FFT* constrói sua aritmética sobre um elemento do corpo complexo, a *NTT* propõe uma abordagem diferente e faz uso de um elemento de um corpo finito. Dessa forma, a aplicação da *NTT* em contextos onde se utilize operações sobre inteiros, como é o caso do YASHE, evita custos relacionados a conversão dos operandos e erros de precisão por conta da aritmética de ponto flutuante da *FFT*, mantendo a redução de complexidade.

3. Implementação

A biblioteca CUYASHE foi desenvolvida durante a execução deste trabalho. Ela foi escrita em C++; utiliza paralelismo sobre CUDA para acelerar a aritmética polinomial; *CRT* para simplificar a manipulação dos operandos na *GPU*; e oferece uma implementação comparativa das transformadas *FFT* e *NTT* na redução da complexidade da multiplicação polinomial. Seu código está disponível ao público sob uma licença GNU GPLv3 [Alves and Aranha 2016c].

A aplicação do *CRT* nos operandos da CUYASHE é feita completamente na *GPU*. Dessa forma, uma vez que os resíduos ainda não foram calculados, o suporte à aritmética de multi-precisão nesse processador torna-se necessário. A biblioteca RELIC [Aranha and Gouvêa 2009] foi utilizada como base e as rotinas requeridas (adição, subtração,

multiplicação, divisão e resto modular) foram adaptadas para CUDA. Enquanto isso, a manipulação de inteiros multi-precisão pela *CPU* (por exemplo, na inicialização e cópia dos dados para a memória global da placa gráfica) foi implementada sobre a biblioteca *NTL* [Shoup 2003].

A implementação da *FFT* foi realizada sobre a biblioteca CUFFT [NVIDIA 2015]. Para isso, precisou-se aplicar operações de conversão dos coeficientes entre o conjunto dos inteiros e o corpo dos complexos. Os erros de precisão oriundos da aritmética de ponto flutuante foram mitigados por meio da redução do tamanho dos coprimos utilizados na decomposição do *CRT*.

Como não se encontrou biblioteca semelhante para a *NTT*, foi necessário que se projetasse e implementasse um algoritmo próprio para essa transformada. A formulação de Stockham para raios 2 e 4 foi utilizada como base [Govindaraju et al. 2008]. Para aplicar a *NTT* de raio R em uma sequência de N elementos, o algoritmo faz uso de $\log_R N$ iterações. Em cada iteração, a transformada é aplicada em R subsequências de tamanho N_s, começando com $N_s = 1$. Ao final, define-se $N_s = RN_s$ e uma nova iteração é iniciada até que $N_s = N$.

Para maximizar os ganhos com o uso das estratégias descritas, a CUYASHE trabalha com resíduos dentro do domínio da transformada. Ou seja, aplica-se o *CRT* seguido da *FFT* ou *NTT* em cada um dos polinômios residuais. Assim, as operações polinomiais de adição e multiplicação têm aplicação coeficiente-a-coeficiente, o que implica em complexidade linear. Além disso, os resíduos são mantidos exclusivamente na memória da *GPU*, dispensando os custos envolvidos com a cópia de dados entre as memórias.

Por fim, desejou-se que a amostragem de distribuições probabilísticas requeridas pelo criptossistema, estreita e Gaussiana discreta, fosse executada exclusivamente na *GPU*, o que elimina o custo de cópia de dados entre as memórias. Para isso, a implementação dessas distribuições foi feita utilizando a biblioteca CURAND [NVIDIA 2015].

4. Resultados

Com o intuito de avaliar a qualidade da implementação e das estratégias utilizadas na CUYASHE, foram tomados quatro trabalhos no estado da arte: Bos *et al.* (BLLN) [Bos et al. 2013]; a implementação de Lepoint e Naehrig (*LN*) [Lepoint and Naehrig 2014]; a biblioteca de Dowlin *et al.* (SEAL) [Dowlin et al. 2015]; e o trabalho de Pöppelmann *et al.* (*PNPM*) [Pöppelmann et al. 2015]. Os três primeiros são baseados em *CPUs*, enquanto o último apresenta uma implementação em *FPGAs*. Foram utilizados os parâmetros de configuração do YASHE propostos por Bos *et al.* como padrão para as comparações de tempo de execução[1]. Lepoint e Naehrig argumentam que esses parâmetros oferecem um nível de segurança de 80 *bits*.

As implementações *LN* e SEAL foram disponibilizadas por seus autores à comunidade. Dessa forma, os tempos de execução das principais operações do YASHE (cifração, decifração, adição e multiplicação homomórfica) apresentados para essas implementações, assim como para a CUYASHE, foram medidos em uma mesma máquina portando uma *CPU Intel Xeon E5-2630 @ 2,6GHz* com uma *GPU NVIDIA GeForce GTX TITAN Black @ 0,98 GHz*. Além disso, utiliza-se a versão 7.5 do kit de desenvolvimento CUDA; a versão 4.8.4 do compilador *g++*; e empregou-se a biblioteca *NTL* 9.1.0 compilada sobre a *GMP*

[1] $R = \mathbb{Z}[X] / \left(x^{4096} + 1\right)$, $q = 2^{127} - 1$, $w = 2^{32}$, $t = 2^{10}$.

6.0.0 [Shoup 2003, Granlund 2012]. Os resultados apresentados são os tempos médios calculados a partir de 100 execuções isoladas de cada operação. A comparação com a BLLN, por outro lado, precisou ser feita exclusivamente por meio dos resultados oferecidos pelos autores, uma vez que seu código não é público. Eles afirmam que os tempos foram medidos sobre uma *CPU Intel Core i7-3520 @ 2,8 GHz* [Bos et al. 2013].

Como comparação adicional, desejou-se avaliar operações intermediárias implementadas na CUYASHE. Para isso, tomou-se como referência a biblioteca CUHE, apresentada no trabalho de Dai e Sunar [Dai and Sunar 2015]. Ela faz uso de paralelismo com a plataforma CUDA e propõe uma série de otimizações na implementação da aritmética polinomial de esquemas baseados em reticulados. Apesar de utilizarem criptossistemas diferentes, a CUHE e a CUYASHE compartilham a estratégia de empregar a *NTT*. Dessa forma, decidiu-se desconsiderar os tempos de execução para operações do criptossistema e apenas manter a comparação para essa função.

4.1. Multiplicação polinomial

Duas estratégias foram testadas para a implementação da multiplicação polinomial: *FFT*, implementada pela CUFFT; e *NTT*, implementada com código próprio. A Tabela 1 apresenta uma comparação do tempo necessário para aplicar uma multiplicação polinomial utilizando cada uma delas. Como pode ser visto, a CUFFT se mostrou de 4 até 7 vezes mais rápida.

Tabela 1. Comparação dos tempos necessários para a multiplicação de dois operandos utilizando a CUFFT e a implementação da *NTT* baseada na formulação de Stockham de raio 4. O *CRT* foi executado com primos de 19 *bits* para a CUFFT e 24 para a *NTT*.

Grau	CUFFT (ms)	*NTT* (ms)
1024	0,3	2,16
2048	0,53	2,06
4096	0,9	4,61
8192	1,71	9,05

Apesar disso, quando comparada com a CUHE, a nossa implementação da *NTT* apresentou ganhos consideráveis, como visto na Tabela 2.

Tabela 2. Comparação dos tempos de execução para *NTT* com raio 4 entre as bibliotecas CUYASHE e CUHE. Os parâmetros de execução são definidos por Dai-Sunar onde os primos usados pelo *CRT* possuem 24 *bits* e são usados respectivamente 15, 25 e 40 polinômios residuais para anéis de grau 8.192, 16.384 e 32.768. Os tempos apresentados para a CUHE foram fornecidos pelos autores em seu trabalho e medidos em uma *GPU NVIDIA GeForce GTX690 @ 1,020GHz*.

Biblioteca	Grau	*NTT* (ms)
CUYASHE	8.192	0,12
CUHE	8.192	0,84
CUYASHE	16.384	0,51
CUHE	16.384	1,78

A *NTT* implementada pela CUHE é uma versão iterativa do algoritmo de Cooley-Tukey [Cooley and Tukey 1965], adaptada para aritmética de corpos finitos. Em relação

a CUHE, a CUYASHE apresenta ganho de 7 vezes em um anel de grau 8.192 e 3,5 vezes em um anel de grau 16.384. Essas resultados sugerem que, apesar de não ter sido capaz de trazer ganho de velocidade em comparação com a CUFFT, a *NTT* implementada neste trabalho tem desempenho compatível ou até mesmo superior ao estado da arte.

A maior velocidade da CUFFT implicou em seu uso como estratégia padrão para multiplicação polinomial na CUYASHE, e dessa forma foi utilizada na obtenção dos resultados apresentados adiante.

4.2. Operações do YASHE

Como pode ser visto na Tabela 3, as operações de cifração, decifração e multiplicação homomórfica da CUYASHE apresentaram tempos 8, 7 e 9 vezes menores do que o trabalho *LN* e 15, 6 e 2, 5 vezes menores do que o BLLN, respectivamente. Sobre a SEAL os ganhos foram ainda mais expressivos, atingindo 19, 17 e 35 vezes, respectivamente. Esses resultados demonstram a influência dos ganhos de velocidade na aritmética polinomial da CUYASHE, principalmente em relação a multiplicações uma vez que as operações do criptossistema são fortemente dependentes dela.

Tabela 3. Comparação entre a CUYASHE e as implementações *LN*, SEAL e BLLN. Os valores para as três primeiras foram medidos na mesma máquina, enquanto a última teve seus tempos fornecidos pelos autores em uma máquina similar. Os tempos para multiplicação homomórfica incluem o custo de relinearização.

Operação	CUYASHE(ms)	*LN*(ms)	SEAL(ms)	BLLN(ms)
Cifração	1,85	15,4	34,93	27
Decifração	2,01	13,71	34,1	5
Adição Homomórfica	0,02	0,59	0,18	0,02
Multiplicação Homomórfica	5,49	31,07	194,94	31

A Tabela 4 compara as operações homomórficas da *PNPM* com a CUYASHE. Apesar da adição homomórfica não ser tão eficiente, a multiplicação se mostra consideravelmente próxima, o que demonstra que a implementação do YASHE em *FPGA* pode ser feita de maneira tão eficiente quanto em *GPGPU*.

Tabela 4. Tempos para o CUYASHE e comparação com os resultados fornecidos pelo trabalho de *PNPM* [Pöppelmann et al. 2015]. Os tempos para multiplicação homomórfica incluem o custo de relinearização.

Operação	CUYASHE(ms)	*PNPM* (ms)
Adição Homomórfica	0,02	0,19
Multiplicação Homomórfica	5,49	6,75

O principal motivo para a utilização de um criptossistema homomórfico é a operação sobre criptogramas. Dessa forma, apesar de ganhos na cifração e decifração serem importantes, operações homomórficas são o alvo principal para otimizações. Assim, os expressivos ganhos de velocidade apresentados sobre o estado da arte têm grande importância na viabilização do criptossistema em uma aplicação real.

A revisão na máquina de estados da CUYASHE em relação a resultados preliminares [Alves and Aranha 2016c, Alves and Aranha 2015b] implicou em uma considerável

melhora nos tempos de execução. A versão atual permite a implementação da aritmética com operandos não apenas decompostos em resíduos pelo *CRT* como também interpolados para o domínio da *FFT* ou *NTT*. Assim, a complexidade computacional dessas operações se tornou majoritariamente linear e evitou-se o custo de aplicação da transformada. Além disso, toda a aritmética passou a ser computada exclusivamente na *GPU*, restando à *CPU* apenas o gerenciamento das operações.

5. Conclusão

Este trabalho investigou estratégias para a implementação do criptossistema YASHE em *GPGPUs* por meio da plataforma CUDA. Com os resultados obtidos, a biblioteca CUYASHE foi desenvolvida, otimizada e disponibilizada à comunidade [Alves and Aranha 2016c].

Aplicou-se o *CRT* para simplificar a manipulação de inteiros multi-precisão na *GPU* e a *FFT* para a redução da complexidade de operações de multiplicação polinomial, das quais o YASHE é altamente dependente. Além disso, realizou-se a implementação comparativa entre as transformadas *FFT* e *NTT*. A primeira foi fornecida pela biblioteca CUFFT, enquanto a *NTT* foi implementada com código próprio baseado na formulação de Stockham. Este trabalho não conseguiu gerar uma implementação da *NTT* que se equiparasse em velocidade com a CUFFT.

A comparação com outros trabalhos da literatura demonstrou ganhos expressivos de velocidade, atingindo uma redução de até 35 vezes no tempo de execução de uma operação de multiplicação homomórfica em relação ao estado da arte, o que sugere que a metodologia proposta foi adequada ao contexto.

Como trabalho futuro, espera-se a avaliação do desempenho da CUYASHE ao ser executada com parâmetros consideravelmente maiores e que evitem o ataque recentemente proposto por Albrecht *et al.* [Albrecht et al. 2016]. Tal ataque tem causado dúvidas na comunidade quanto a viabilidade do uso do YASHE em aplicações reais, como em algoritmos de análise de dados e aprendizagem de máquina que requerem considerável profundidade multiplicativa.

Referências

[Albrecht et al. 2016] Albrecht, M., Ducas, L., and Bai, S. (2016). A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes. In *CRYPTO 2016*.

[Alves and Aranha 2015a] Alves, P. and Aranha, D. (2015a). cuYASHE: Computação sobre dados cifrados em GPGPUs. In *X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica em Andamento do IC-Unicamp*, pages 198–210.

[Alves and Aranha 2015b] Alves, P. and Aranha, D. (2015b). cuYASHE: Computação sobre dados cifrados em GPGPUs. In *XV Simpósio Brasileiro de Segurança da Informação e Sistemas Computacionais (SBSeg 2015)*, pages 55–60.

[Alves and Aranha 2016a] Alves, P. and Aranha, D. (2016a). Computação sobre dados cifrados em GPGPUs. Dissertação de mestrado, Instituto de Computação da Universidade Estadual de Campinas, Campinas, SP, Brasil.

[Alves and Aranha 2016b] Alves, P. and Aranha, D. (2016b). Computação sobre dados cifrados em GPGPUs. In *Anais do XXXVI Congresso da Sociedade Brasileira de Computação - I ETC*, pages 816–819.

[Alves and Aranha 2016c] Alves, P. and Aranha, D. (2016c). cuYASHE. `https://github.com/cuyashe-library/cuyashe`. Último acesso: 13/07/2016.

[Aranha and Gouvêa 2009] Aranha, D. F. and Gouvêa, C. P. L. (2009). RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

[Bos et al. 2013] Bos, J., Lauter, K., Loftus, J., and Naehrig, M. (2013). Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Stam, M., editor, *Cryptography and Coding*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer Berlin Heidelberg.

[Cooley and Tukey 1965] Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301.

[Dai and Sunar 2015] Dai, W. and Sunar, B. (2015). cuHE: A Homomorphic Encryption Accelerator Library. Cryptology ePrint Archive, Report 2015/818.

[Dinh et al. 2013] Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611.

[Dowlin et al. 2015] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2015). Manual for Using Homomorphic Encryption for Bioinformatics.

[Govindaraju et al. 2008] Govindaraju, N. K., Lloyd, B., Dotsenko, Y., Smith, B., and Manferdelli, J. (2008). High Performance Discrete Fourier Transforms on Graphics Processors. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA. IEEE Press.

[Granlund 2012] Granlund, T. (2012). *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition. Último acesso: 05/03/2016.

[Hoffa et al. 2008] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., and Good, J. (2008). On the Use of Cloud Computing for Scientific Workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645.

[Lepoint and Naehrig 2014] Lepoint, T. and Naehrig, M. (2014). A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Progress in Cryptology – AFRICACRYPT 2014*, volume 8469, pages 318–335. Springer International Publishing.

[Lindner and Peikert 2011] Lindner, R. and Peikert, C. (2011). *Better Key Sizes (and Attacks) for LWE-Based Encryption*, pages 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg.

[NVIDIA 2015] NVIDIA (2015). CUDA Toolkit Documentation. Último acesso: 23/03/2016.

[Pöppelmann et al. 2015] Pöppelmann, T., Naehrig, M., Putnam, A., and Macias, A. (2015). *Accelerating Homomorphic Evaluation on Reconfigurable Hardware*. Springer Berlin Heidelberg.

[Shoup 2003] Shoup, V. (2003). NTL: A library for doing number theory. `http://www.shoup.net/ntl`. Último acesso: 05/03/2016.

[Xiao and Xiao 2013] Xiao, Z. and Xiao, Y. (2013). Security and Privacy in Cloud Computing. *IEEE Communications Surveys Tutorials*, 15(2):843–859.