PAULO CESAR DE FIGUEIREDO CRUZ

UMA NOVA ABORDAGEM DA PROGRAMAÇÃO PARALELA PARA GPU UTILIZANDO PLATAFORMAS DE DESENVOLVIMENTO EM SERVIÇOS DE NUVEM

Limeira
2014

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

PAULO CESAR DE FIGUEIREDO CRUZ

UMA NOVA ABORDAGEM DA PROGRAMAÇÃO PARALELA PARA GPU UTILIZANDO PLATAFORMAS DE DESENVOLVIMENTO EM SERVIÇOS DE NUVEM

Trabalho de conclusão de curso submetido à Universidade Estadual de Campinas, como parte dos requisitos obrigatórios para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Vitor Rafael Coluci

Limeira
2014

Dissertação de trabalho de conclusão de curso

Uma nova abordagem na programação paralela para gpu utilizando plataformas de desenvolvimento em serviços de nuvem

Autor: Paulo Cesar de Figueiredo Cruz

Orientador: Prof. Dr. Vitor Rafael Coluci

Data: 18 de Junho de 2014

A banca examinadora composta pelos membros abaixo aprovou esta dissertação

_____

Prof. Vitor Rafael Coluci, Dr
FT/UNICAMP

_____

Prof. Ana Estela Antunes da Silva, Dra
FT/UNICAMP

_____

Ana Maria de Seixas Pereira, Ma
CENAPAD/UNICAMP

Limeira
2014

RESUMO

Com o passar dos anos o poder de processamento dos computadores cresceu. No entanto, mesmo com a contínua produção de novos *chips* com maior poder de processamento, novas abordagens que utilizam o conceito do paralelismo têm sido propostas para reduzir o tempo de processamento. Uma dessas propostas é a junção do conceito do paralelismo ao sistema de nuvens computacionais, visando permitir execução paralela com característica altamente distribuída. Nesse sentido, neste trabalho, desenvolvemos um sistema CloudLU, capaz de integrar os conceitos de computação na nuvem com o de distribuição de dados para processamento paralelo em um ou vários nós computacionais que podem estar dispersos geograficamente. Para demonstração de conceitos, o sistema CloudLU foi desenvolvido baseado na plataforma de desenvolvimento Microsoft .NET, através de uma aplicação para decomposição LU de matrizes em unidades de processamento gráfico (GPU) interligada a uma aplicação também baseada na mesma plataforma, para os serviços de nuvem Windows Azure. Com este sistema, se verificou que a distribuição facilitada pela nuvem e o processamento via GPU permitiu ganhos de até 7 vezes em relação ao processamento sequencial local.

Palavras-chave: Paralelismo, Computação Distribuída, Nuvem, Plataforma de desenvolvimento.

ABSTRACT

Over the years the processing power of computers has grown. However, even with the continuous production of new chips with more processing power, new approaches that use the concept of parallelism have been proposed to reduce the processing time. One such approach is the junction the concept of parallelism to the cloud computing system, in order to enable parallel execution with highly distributed feature. Accordingly, in this work, we developed the CloudLU system capable to incorporate the concepts of cloud computing with the distribution of data for parallel processing on one or more computational nodes that can be geographically dispersed. For demonstration of concepts, the CloudLU system was developed based on the Microsoft development framework. NET, through an application for LU decomposition of matrices on graphics processing units (GPU) also linked to an application based on the same framework for the Windows Azure cloud services. With this systemit has been found that the distribution of the cloud facilitated and enabled GPU processing gain of up to 7 times compared to local sequential processing.

Key words: Parallel, Distributed Computing, Cloud, Development frameworks.

SUMÁRIO

Dedico este trabalho inicialmente à Deus e em especial à minha querida filha Patrícia.

AGRADECIMENTOS

Durante o tempo dedicado ao estudo de tecnologias, tive o prazer de obter muitas conquistas, dentre elas a motivação necessária para realizar o sonho de estudar em uma conceituada universidade, como esta. Para tanto é necessário fazer os devidos agradecimentos, pois por fim, chego ao final de mais um ciclo com muita gratidão.

Agradeço:

A Deus por iluminar meu caminho, amparando e confortando nos momentos de tropeços.

A minha querida avó Odete a qual, desde pequeno meu deu muito amor e carinho, ao meu avô Paulo, que como um pai, esteve presente em momentos importantes da minha vida e também a minha irmã Daniela, amiga e companheira.

A minha amada filha Patrícia, que em determinados momentos foi privada da minha presença física e a quem eu dedico todo o esforço para chegar até aqui.

Aos meus amigos e companheiros de faculdade, os quais tive o prazer de conviver durante todo este tempo.

A todas as pessoas especiais, que me acompanharam, me acompanham nos dias atuais e que me acompanharão para o resto da vida.

A todos os professores, merecedores de todos os créditos pela dedicação à educação.

Ao meu orientador Prof. Dr. Vitor Rafael Coluci, que me orientou no momento propício, dando apoio e atenção.

E a Universidade Estadual de Campinas, por propiciar uma excelente formação acadêmica e uma grande oportunidade de participar de um intercambio internacional.

A todos o meu muito OBRIGADO.

"Penso, logo existo."

René Descartes

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

BLOB - *Binary Large Object*

CPU - *Central Processing Unit*

CUDA - *Compute Unified Device Architecture*

ECMA - *European Computer Manufacturers Association*

GPU - *Graphics Processing Unit*

GPGPU - *General Purpose Graphics Processing Unit*

JSON - *Java Script Object Notation*

LU - *Lower Upper*

MVC - *Model, View, Controller*

REST - *Representational State Transfer*

SQL - *Structured Query Language*

1. INTRODUÇÃO

Com a massificação dos computadores de uso pessoal e a evolução do acesso a Internet, tanto no meio acadêmico, como corporativos, é possível notar que a demanda por novos aplicativos cresce de acordo com as necessidades intrínsecas de cada pessoa ou organização.

Com isto surge a necessidade de novas abordagens acerca de como processar informações de maneira rápida e segura. O paralelismo de processamento em conjunto com a distribuição oferece ferramentas para desenvolvimento de uma nova abordagem.

De acordo com a analogia feita por (Patterson e Hennessy, 1998) "oito repórteres tentando escrever uma história única, na esperança de fazer o trabalho oito vezes mais rápido", fornece a idéia central do uso do paralelismo.

Ocorre que, de acordo com a analogia apresentada, se o trabalho for executado apenas por um repórter, o trabalho poderá ser concluído, porém em um tempo maior. O mesmo ocorre ao processar informação de maneira sequencial, ou seja, o tempo é maior e o resultado é o mesmo.

Considerando os tempos de processamento de uma aplicação estruturada para execução seqüencial leva em um determinado computador, e a mesma aplicação estruturada para processamento paralelo e distribuído, podemos então, fazer uma análise visando a melhoria que pode ser obtida utilizando estas duas tecnologias (o paralelismo, e a distribuição de informação) aplicada a um estudo de caso.

Daí deriva-se a pergunta deste trabalho “Como é possível utilizar a programação paralela unida à programação distribuída para que juntas obtenham maior ganho em relação ao processamento sequencial?”.

Esta pergunta pode ser feita com relação a dados pré-existentes nas organizações, acumulados durante determinado tempo, ou até mesmo dados produzidos em tempo real e que necessitem de maior rapidez em seu processamento.

A utilização de unidades de processamento gráfico (GPU, do inglês, *Graphics Processing Unit*) com o intuito de desenvolver aplicações de uso geral vem sendo apreciada desde o momento em que a utilização do processamento através de placas gráficas se tornou massivo.

De acordo com a empresa NVIDIA que produziu os primeiros chips GPU:

> Os chips gráficos começaram como processadores de vídeo com função fixa, mas se tornaram cada vez mais programáveis e poderosos em termos de computação, o que levou a NVIDIA a lançar a primeira GPU. No período entre 1999 e 2000, cientistas de computação e cientistas de campo em diversas áreas começaram a usar GPUs para acelerar uma variedade de aplicativos científicos. Esse foi o início do movimento chamado de GPGPU, ou Computação de Uso Geral na GPU (NVIDIA, 2014)

Levando em consideração esse avanço no uso geral das GPU, a empresa Nvidia desenvolveu o modelo de programação paralela CUDA (Compute Unified Device Architecture), no qual este projeto se baseia para prover uma solução de processamento rápido e eficiente.

Na mesma linha da utilização da GPU para diversos propósitos, outro conceito semelhante ocorre com a utilização de serviços na nuvem. Inicialmente concentrados de acordo com a necessidade de cada organização, hoje são amplamente disponibilizado provendo grande variedade de serviços, também adaptados a praticamente qualquer propósito.

Pois, de acordo com (TULLOCH, 2013) "como uma plataforma da Microsoft que provê um range de diferentes serviços, Windows Azure permite construir, implantar e gerenciar soluções para praticamente qualquer propósito que imaginar".

Deste modo, a utilização da computação de uso Geral GPGPU, e os serviços de nuvem disponibilizados pela Microsoft através do Windows Azure, pode permitir aproveitar a capacidade e distribuição de processamento ocioso existente na maioria dos computadores domésticos ou corporativos.

1.1 Objetivos

Desta forma, neste trabalho propomos uma aplicação distribuída, com a utilização dos serviços em nuvem, baseados na plataforma de desenvolvimento Microsoft .NET que possa disponibilizar dados para que sejam processados em nós separados geograficamente. Além disso o paralelismo seria oferecido por GPU´s localizadas nos nós computacionais, com suas respectivas aplicações a fim de obter ganhos de desempenho em relação ao processamento sequencial.

1.2 Organização textual

No Capítulo 2, apresentamos uma revisão do tema da programação paralela para GPU, que será parte do sistema CloudLU que será mostrado nos capítulos seguintes.

No Capítulo 3, apresentamos uma revisão da literatura referente ao tema de computação em nuvem utilizando o serviço da Microsoft, Windows Azure.

No Capítulo 4, descrevemos a metodologia utilizada neste projeto, evidenciando as partes que o compõem e as métricas utilizadas para quantificação do ganho de desempenho computacional. Apresentamos ainda a decomposição LU, utilizada como estudo de caso na aplicação do sistema CloudLU desenvolvido.

No Capítulo 5, descrevemos o sistema CloudLU desenvolvido, que engloba vários conceitos apresentados nos capítulos anteriores, unindo a capacidade de processamento que é obtida através da utilização do modelo CUDA em GPU, com a computação em nuvem.

No Capítulo 6, apresentamos os resultados obtidos e, finalmente no Capítulo 7 apresentamos as conclusões do trabalho e considerações finais.

2. PROGRAMAÇÃO PARALELA UTILIZANDO GPU

2.1 O que é GPU?

GPU (do inglês *Graphic Processor Unit*), é a unidade de processamento gráfico presente atualmente em muitos computadores e notebooks de uso doméstico e empresarial. “A computação por GPU é o uso da unidade de processamento gráfico, juntamente com a CPU (do inglês, *Central Process Unit*) para acelerar aplicações de engenharia, ciências e de organizações.” (NVIDIA, 2014).

> Uma maneira simples de entender a diferença entre uma CPU e GPU é comparar como elas processam tarefas. A CPU consiste de alguns núcleos otimizados para processamento de série sequencial, enquanto uma GPU consiste em milhares de núcleos menores, mais eficientes projetados para lidar com múltiplas tarefas simultaneamente. (NVIDIA, 2014).

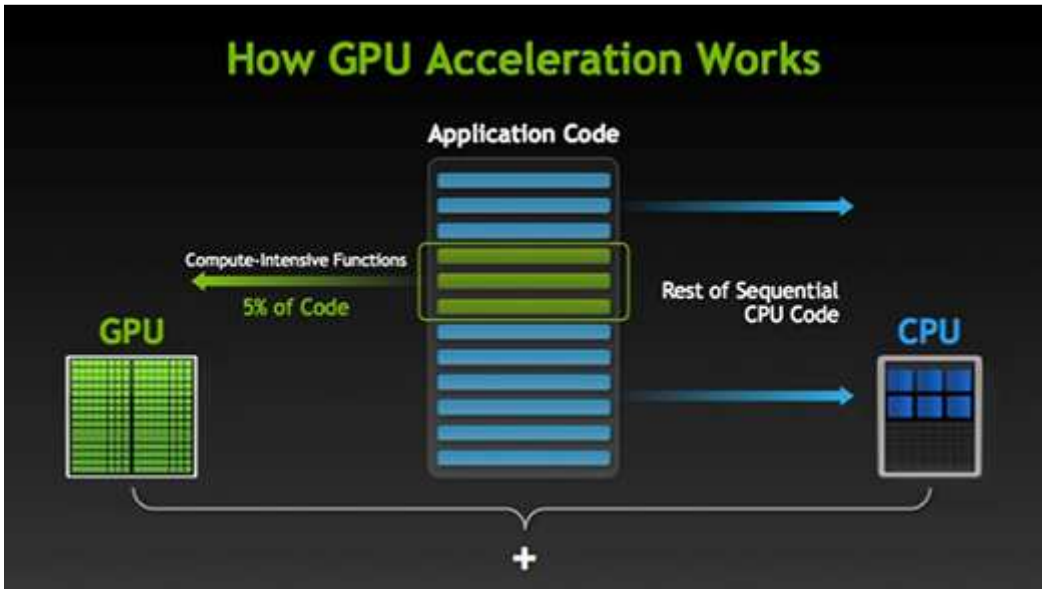

Figura 1 - Como funciona a aceleração por GPU

Fonte: adaptado de (NVIDIA, 2014)

"O termo GPGPU foi cunhado por Mark Harris em 2002 quando ele reconheceu uma tendência inicial do uso da GPU para aplicações não gráficas." (GPGPU.ORG, 2014);

> Nos dias atuais a GPU pode implementar diretamente muitos algoritmos paralelos utilizando placas gráficas. Com isto, é possível afirmar que a programação paralela utilizando GPU é passível de ser programada para qualquer propósito. (LUEBKE e HUMPHREYS, 2007)

Para que isto seja possível é necessário utilizar um modelo de programação capaz de fornecer a interface entre o desenvolvimento de código e a efetiva execução deste código por uma GPU, que é o caso do modelo CUDA.

2.2 O que é CUDA?

CUDA (*Compute Unified Device Architecture*) "é uma plataforma e modelo de programação paralela que permite um aumento dramático no desempenho de computação, aproveitando o poder da unidade de processamento gráfico (GPU)." (NVIDIA, 2014).

O modelo de programação CUDA permite o desenvolvimento de aplicações paralelas utilizando diferentes linguagens de programação, entre elas a C, C++ e C#.NET, que será apresentada nos capítulos seguintes.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Figura 2 - Código de adição de vetores utilizando o modelo CUDA
Fonte: adaptado de (NVIDIA, 2014)

Uma ilustração (Figura 2) do código mostra como é feita a adição de dois vetores A e B de tamanho N e armazenando o resultado em um vetor C, paralelamente através da chamada ao método VecAdd. A implementação do método VecAdd, utiliza a palavra-chave __global__, o qual identifica que este método será executado na GPU e que é chamado através da CPU com a utilização de colchetes triplos (<<< 1, N >>>>) (Figura 2), também conhecido como chamada de núcleo.

3. COMPUTAÇÃO DISTRIBUÍDA COM UTILIZAÇÃO DE NUVEM

> “Windows Azure pode ser o que você quiser que seja.” Como uma plataforma de nuvem da Microsoft que fornece uma gama ampla de diferentes serviços, Windows Azure, permite a você criar, implantar e gerenciar soluções para qualquer finalidade que você imaginar. (TULLOCH, 2013)

A plataforma de nuvem da Microsoft (Figura 3), se baseia em quatro tipos básicos de serviços: Serviços de Computação, de Rede, de Dados e de Aplicações. (TULLOCH, 2013).

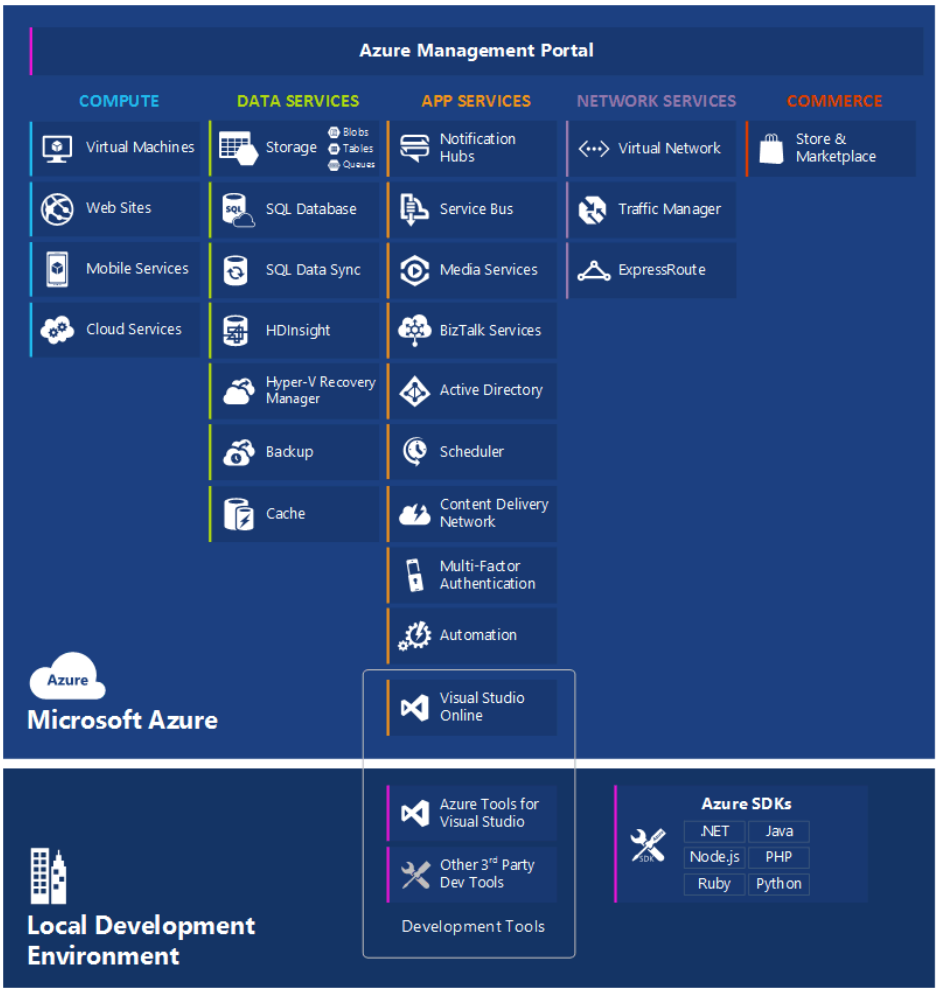

Figura 3 - Características dos serviços de nuvem Windows Azure.

Fonte: adaptado de (MICROSOFT, 2014)

Devido às extensas características que envolvem os serviços do Windows Azure, neste projeto focaremos apenas em dois dos serviços básicos, sendo eles: Serviços de Computação e Serviços de Dados, que contribuirão para a codificação do sistema CloudLU. Uma descrição mais detalhadas desses tipos de serviços é apresentada no APÊNDICE A.

4. METODOLOGIA

O projeto foi realizado baseando-se no desenvolvimento de um sistema CloudLU, que foi aplicado na decomposição LU e identificação dos seguintes algoritmos: um para o processamento sequencial, outro com a utilização de *threads* e outro algoritmo que utiliza o modelo CUDA para processamento em GPU.

Para os testes do sistema, utilizamos um equipamento com uma GPU e acesso a uma conexão com a Internet de alta velocidade para que esta permitisse a comunicação necessária com o sistema de nuvem Windows Azure.

A plataforma de desenvolvimento utilizada foi a Microsoft .NET, com um conjunto de ferramentas e componentes (APÊNDICE B), juntamente com os recursos do Visual C#. Essa linguagem de programação foi escolhida por sua facilidade de uso e também por experiência profissional deste autor, além da existência de extensas referências disponíveis para consulta em fóruns e sites de tecnologia.

Utilizando a plataforma .NET, inicialmente os algoritmos pesquisados foram implementados e executados mediante testes individuais com a finalidade de estabelecer métricas básicas que, na análise de resultados, foram utilizadas para comparar com as métricas extraídas do sistema CloudLU desenvolvido.

Após as análises e medições iniciais, desenvolvemos o sistema CloudLU proposto, composto pelo sistema distribuído baseado nos serviços disponibilizados em nuvem através do Windows Azure, utilizado para gerenciar as informações e o sistema nos nós de processamento.

Realizamos testes do sistema CloudLU aplicando-o em matrizes de diferentes tamanhos.

Ao final foram avaliados os resultados obtidos com o sistema CloudLU, comparando os resultados de sua execução com a execução do algoritmo sequencial estudado, através da medição do *speedup*, métrica evidenciada por (AMDAHL 1967) que visa estabelecer o ganho que pode ser obtido por meio do algoritmo paralelo em relação ao sequencial.

A métrica do *speedup*, neste projeto, teve seu conceito estendido com a adição de tempos relacionado à movimentação de dados, ou seja, à leitura e gravação de dados. Isso se deve ao fato de que, para o sistema distribuído, estes tempos impactam no tempo total de processamento, necessário para a medição e comparação do ganho obtido.

4.1 A decomposição LU como estudo de caso

A decomposição LU é uma das técnicas mais usadas para resolver sistemas de equações algébricas e consiste em dividir a matriz **A** em duas submatrizes triangulares: **L** (de *lower*, inferior em inglês) e **U** (de *upper*, superior em inglês).

Além disso, "o método usado no benchmark LINPACK é a decomposição LU com pivotamento parcial, o qual serve como medida para elaboração da lista dos 500 computadores mais rápidos do mundo." (TOP500, 2014).

> O pacote LINPACK é uma coleção de sub-rotinas Fortran para resolução de vários sistemas de equações lineares. Este pacote é baseado na decomposição numérica da álgebra linear.A idéia geral é que, dado um problema que envolve uma matriz A, um fator a decompõe num produto de matrizes simples, bem estruturados, que podem ser facilmente manipulados para resolver o problema original.
> (DONGARRA, LUSZCZEK e PETIT, 2003)

Todos os algoritmos aqui pesquisados levam em consideração a decomposição LU, com pivoteamento parcial assemelhando-se á utilizada no LINPACK.

> Dentre as técnicas para melhora no desempenho na resolução dos sistemas lineares, uma das que se destacam no LINPACK é o reuso dos dados, sabe-se que a cada passo do processo de fatoração do LINPACK são feitas operações vetoriais para modificar uma submatriz inteira dos dados. Essa atualização faz com que um bloco dos dados seja lido, atualizado e escrito novamente na memória central. Uma possível solução é reestruturar os algoritmos de modo que explorem a hierarquia de memória das arquiteturas, o que pode ser feito, por exemplo, armazenando os dados o maior tempo possível nas memórias de nível mais próximo do processador. (MILANI, 2008)

Para o caso em que iremos processar uma decomposição LU com a utilização de CUDA, todo o conjunto que compõe a matriz de elementos foi armazenada na memória da GPU, ou seja, a memória mais próxima do processador (MILANI, 2008). Já para o algoritmo sequencial, a decomposição LU foi feita de maneira tradicional, ou seja, processando o bloco normalmente e transferindo-o para a memória principal.

Ainda em relação ao reuso dos dados, todas as operações feitas na matriz de elementos foram executadas e devolvidas ao mesmo destino de memória de origem, visando economia de memória e evitando a movimentação de dados. Este caso de reuso foi utilizado em todos os algoritmos aqui pesquisados, resultando sempre em uma matriz final **A'** que contém em sua estrutura triangular superior o equivalente à matriz **U** em em sua estrutura triangular inferior, o equivalente à matriz **L**.

4.2 Equipamento utilizado

- Notebook Dell Vostro 3300
- Cpu Intel Core I5 M560 2.67GHz, 4 núcleos de processamento
- Memória DDR3 8Gb
- Disco Rígido Primário: 120GB Kingston SSD Serial Ata
- Disco Rígido Secundário: 500GB Samsung Serial Ata
- Placa de Vídeo Primária: Intel HD Graphics
- Placa de Vídeo Secundária: NVIDIA GeForce 310M, memória de 512mb, compatível com CUDA 1.2
- Sistema Operacional Microsoft Windows 7 Pro x64

Conexão com a internet banda larga

- Tipo: Conexão Wifi
- Local: Universidade Estadual de Campinas – Campus Limeira
- Taxa de download medida em 08/05/2014: 26.52 Mbps
- Taxa de upload medida em 08/05/2014: 20.83 Mbps

Limitações do Equipamento

O equipamento utilizado possui uma GPU Nvidia 310M, compatível com Cuda 1.2 e com memória de 512MB, sendo estes dois itens fatores limitantes para este projeto. A memória de 512MB, limita o espaço disponível para o armazenamento dos dados de uma matriz. Portanto, ao se estabelecer os critérios de testes, este fator contribuiu para limitar o tamanho da matriz estudada de acordo com o tamanho desta memória.

De acordo com (WILT, 2013) o suporte a precisão dupla foi inserida apenas a partir da versão 1.3 nas GPUs compatíveis com CUDA. Sendo assim, devido à limitação do hardware, a precisão adotada foi a de ponto flutuante simples, a qual limita os dígitos de precisão em até 7 casas decimais, de acordo com o descrito pela norma IEEE 754 (ANSI/IEEE, 1982).

Levando em consideração as limitações impostas, o tamanho máximo que pode ser ocupado de memória é 512MB que representa 4.294.967.296 bits, em ponto flutuante de precisão simples.

Seja **T**(n), a matriz quadrada n x n em precisão simples em ponto flutuante que uma determinada porção de memória M em bits pode ocupar, e M2 a quantidade de bits que um

determinado numero em precisão simples ocupa desta memória. A equação 1, estabelece a maneira para calcular a ordem máxima que uma matriz pode ter na memória de uma GPU.

$$T(n) = \sqrt{\frac{M}{M2}} \tag{1}$$

Portanto, ao dividir 4.294.967.296 por 32bits, resultando em 134.217.728 números em ponto flutuante de precisão simples, extraindo a raiz quadrado deste ultimo, se pode obter a ordem máxima que uma matriz na memória da GPU pode ocupar é n = 11.585, considerando apenas o número inteiro de elementos.

Porém, é necessário considerar que a memória da GPU não pode ser inteiramente ocupada com dados. É necessário, por exemplo, espaço para utilização de variáveis auxiliares de troca, entre outros elementos necessários a programação em CUDA. Devido a isso, estabelecemos a ordem máxima de n = 10.000.

5. SISTEMA CLOUDLU

O objetivo geral do sistema é disponibilizar uma interface para o usuário, o qual poderá submeter arquivos de dados contendo uma matriz a ser decomposta em triangular superior e inferior. Esse sistema, por meio da nuvem, distribuirá esses arquivos entre os nós de processamento. Os nós de processamento utilizarão um aplicativo de processamento para a decomposição LU utilizando GPU com o modelo de programação CUDA. Uma vez finalizado o processamento nos nós, o sistema retornará os resultados ao sistema na nuvem.

Com o intuito de estabelecer métricas de comparação para os dados processados, efetuamos execuções sequenciais dos algoritmos de decomposição LU para posteriormente, executar a mesma decomposição LU no sistema CloudLU baseado em serviços de nuvem.

5.1 Arquitetura do sistema

Elucidados alguns dos principais componentes que compõem o sistema CloudLU nos capítulos anteriores, vamos agora fazer uma apresentação deste sistema, através do fluxo da aplicação em 12 passos (Figura 4), primeiramente descrevendo os componentes principais da aplicação, sendo eles, o nó de gerenciamento, o nó de processamento e os componentes de nuvem(serviços de computação e de dados).

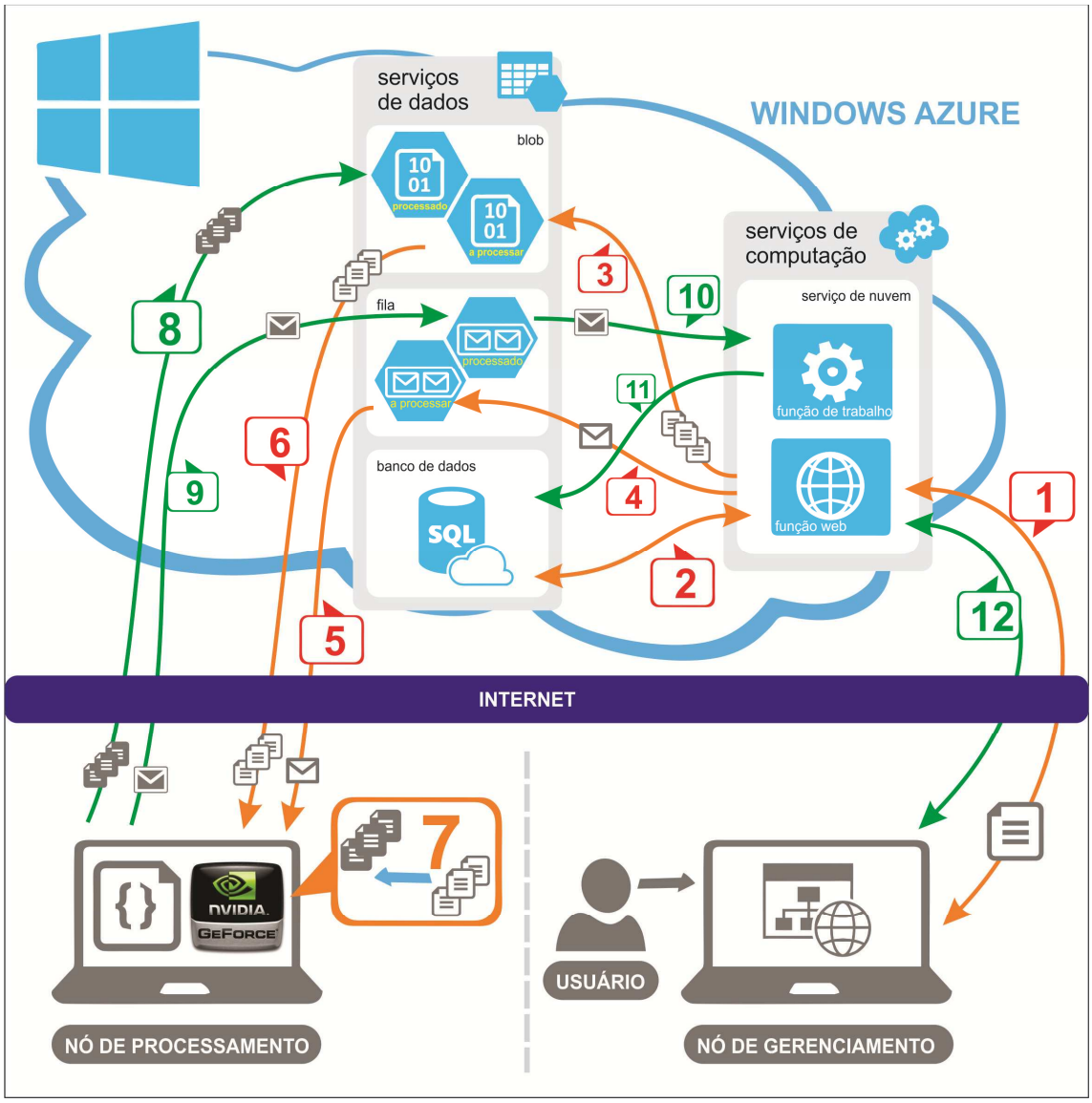

Figura 4 - Arquitetura do sistema CloudLU

A estrutura completa da solução foi estruturada no aplicativo Visual Studio e está descrita no APÊNDICE D, compatível com a plataforma .NET (APÊNDICE B),.

Os componentes de nuvem ou serviços de nuvem que abordamos neste sistema compreendem:

- Serviços de Dados, responsáveis principalmente pelo armazenamento dos dados do projeto, composto por duas filas (uma de dados a processar e outra de dados processados), um banco de dados Sql Server e os recipientes de blob onde serão armazenados os arquivos principais de uma decomposição LU e outro recipiente blob para seus respectivos resultados.

- Serviços de computação, compreendendo as funções web e de trabalho, como segue:

- A função web hospeda o serviço web, e é composto por uma aplicação desenvolvida no padrão ASP.NET MVC (*Model*, *View*, *Controller*), a qual ficará disponível para os usuários acessarem através de um ou mais nós de gerenciamento (APÊNDICE E). Composta por 4 páginas principais: o projeto, nova execução, histórico e autor.
- A função de trabalho, neste sistema, é responsável pelo monitoramento da fila de resultados processados e processamento destas mensagens para então compor o histórico da aplicação.

O nó ou nós de gerenciamento, são responsáveis por disparar os eventos necessários ao fluxo de trabalho da aplicação. E são acessados através de uma interface web utilizando navegadores conhecidos, como Internet Explorer, Firefox ou Chrome.

O nó ou nós de processamento (APÊNDICE E) são os responsáveis por monitorar a fila de mensagens a processar, ele deverá possuir as características do equipamento aqui utilizado, contendo em sua configuração física o processador GPU como mandatório. Este equipamento possui o sistema de processamento responsável por receber os dados do sistema em nuvem, processá-los e submeter os resultados.

5.1.1 Fluxo principal da arquitetura

A Figura 4 apresenta o fluxo demarcado na cor laranja representando o fluxo de solicitação de processamento, enquanto que o fluxo demarcado na cor verde, representa o fluxo de resultado deste processamento. inicialmente se seguirá um resumo dos passos elencados nesta, para então detalhá-los a seguir.

O usuário acessa a função web (1), responsável pelo gerenciamento do sistema, abre a página de nova execução, insere os parâmetros da execução, em seguida o sistema envia a informação para o banco de dados (2), armazenando os parâmetros da execução solicitada, a função web envia os dados ao recipiente de itens a processar, que armazenará os arquivos necessários a execução da decomposição LU, finalizado a cópia dos arquivos (3) a função web envia uma mensagem para a fila de mensagens a processar (4), contendo os parâmetros da execução. Um dos nós de processamento monitora a fila de mensagens a processar (5), e detectando a presença de alguma mensagem, visualiza seu conteúdo e inicia o processo de cópia dos arquivos do recipiente elencado em (3), para si (6), compõe-se novamente então, a matriz de elementos a serem processados na memória principal deste nó, executa-se

posteriormente a cópia desta matriz da memória principal para a memória da GPU e dispara-se o processo de decomposição LU (7). Ao finalizar o processo de decomposição LU, o sistema gera então novos arquivos de resultado e os armazena no recipiente do serviços de dados de itens processados (8), ao final deste processo, encaminha uma mensagem com os dados e resumo dos tempos desta execução para a fila de itens processados (9). A função de trabalho por sua vez, monitorando a fila de itens processados, verifica a presença da mensagem,visualiza e processa seu conteúdo (10), enviando seus resultados para o banco de dados (11), onde por fim ficará disponível para que o usuário tenha acesso ao histórico e ao arquivo de resultado final da decomposição LU (12).

Seguindo com os detalhes, é possível observar (Figura 4) que há uma marcação dos passos de um a doze. Estes são os passos que a aplicação segue para distribuir e processar os dados de uma matriz, a qual será feita sua decomposição LU através do processamento em uma GPU de um nó de processamento.

Inicialmente, o usuário acessa a aplicação por um navegador (1), este por sua vez aciona a função web, que está hospedada na nuvem, informa uma nova execução, passando os parâmetros necessários. A página de criação de uma nova execução (Figura 5), contém a tabela de *status* das execuções, esta tabela contém informações sobre a data de solicitação da execução, a ordem da matriz de elementos solicitada e o *status* da execução, se concluído ou pendente.

O usuário neste passo (1) pode selecionar entre uma execução existente, ou seja, dados pré-carregados conforme os testes dos algoritmos de decomposição LU, nas ordens da matriz: 10, 100, 1000, 5000 e 10000 e pode, como alternativa, selecionar um arquivo para *upload*. Este arquivo único, deverá conter os dados pré-formatados de acordo com os parâmetros estabelecidos (APÊNDICE F).

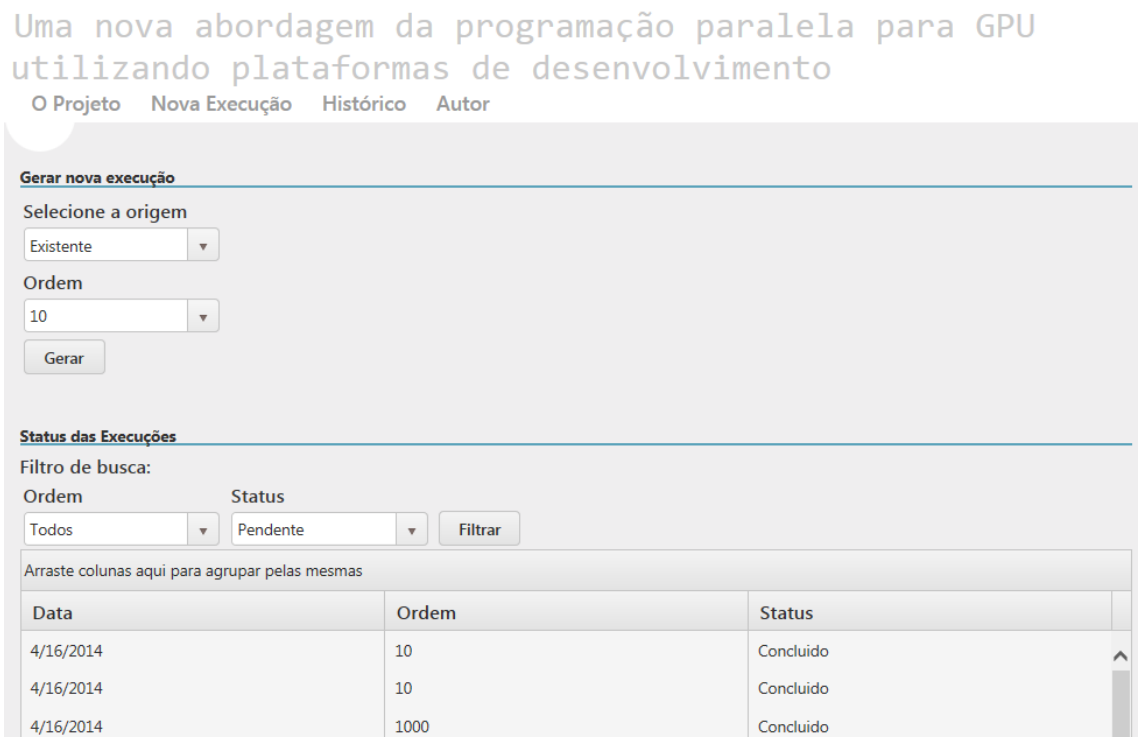

Figura 5 - Página da função web para criar nova execução

Ao gerar a execução (Figura 5) o sistema confirmará se obteve sucesso. Caso contrário, informará o erro ocorrido e então armazenará no banco de dados (2) os dados do projeto de execução, retornando um número de identificação provinda da base de dados (APÊNDICE H) para o projeto criado. Posteriormente o arquivo que o usuário selecionou para *upload* e/ou com a utilização do pré-existente será formatado, compactado e enviado para o recipiente de itens a processar.

Antes de enviar para o recipiente, o arquivo informado para *upload* é dividido em arquivos menores, de acordo com a ordem da matriz. Utiliza então, um processo de compactação para cada arquivo, ou seja, se a ordem da matriz é 10x10, cada arquivo gerado será referente a uma linha desta matriz, contendo 10 elementos. E, ao final do processo 10 arquivos compactados são gerados, com a numeração de 0 a 9 com a extensão .zip, que representa o formato do arquivo no recipiente.

Utilizar um processo de compactação dos arquivos (APÊNDICE G) foi necessário para diminuir o tempo de envio/recebimento deste arquivo, por meio de uma implementação baseada no sistema de threads da plataforma .NET, que utiliza o padrão produtor/consumidor.

Os arquivos gerados são então gravados no formato blob no recipiente (3) com o seguinte padrão de nomenclatura:

"http://SUACONTA.blob.core.windows.net/RECIPIENTE/ID-ORDEM-NUMERO.zip",

Onde SUACONTA representa o nome utilizado para cadastro nos serviços de armazenamento no Windows Azure, RECIPIENTE representa o nome do recipiente onde serão armazenados os arquivos da matriz que foi submetida para processamento, e "ID-ORDEM-NUMERO.zip" representa o nome de cada arquivo submetido, onde ID é a identificação do projeto de execução cadastrado no banco de dados, ORDEM representa a ordem da matriz submetida e NUMERO, representa a parte compactada do arquivo representando um numero de 0 a ORDEM-1 da matriz submetida.

O processo de envio destes arquivos utiliza o modelo REST (APÊNDICE A), com chamadas através do protocolo HTTP, por se tratar de uma maneira rápida de enviar os arquivos.

Finalizado o envio dos arquivos para o recipiente (3), a função web envia uma mensagem para a fila de mensagens a processar (4), contendo os parâmetros da execução.

Esta mensagem enviada para a fila utiliza o modelo REST através do protocolo HTTP como protocolo de conexão. O corpo da mensagem que é enviada para a fila é composto por uma cadeia de caracteres estruturado no padrão JSON. Este padrão foi utilizado por permitir a serialização de uma classe da linguagem Visual C# no formato estruturado de caracteres, para posterior de-serialização no nó de processamento, facilitando e tornando ágil o processo de identificação dos parâmetros para processamento.

O nó de processamento possui o aplicativo de processamento (Figura 6), que monitora a fila de mensagens a processar (5) e, detectando a presença de alguma mensagem, visualiza seu conteúdo e inicia o processo de cópia dos arquivos do recipiente elencado em (3) para si (6).

Ao monitorar a fila de mensagens de itens a processar, e ao detectar uma nova tarefa, faz a cópia dos arquivos do recipiente de itens a processar, compondo então, a matriz de elementos a serem processados na memória principal deste nó. Executa-se posteriormente a cópia desta matriz da memória principal para a memória da GPU e dispara-se o processo de decomposição LU (7).

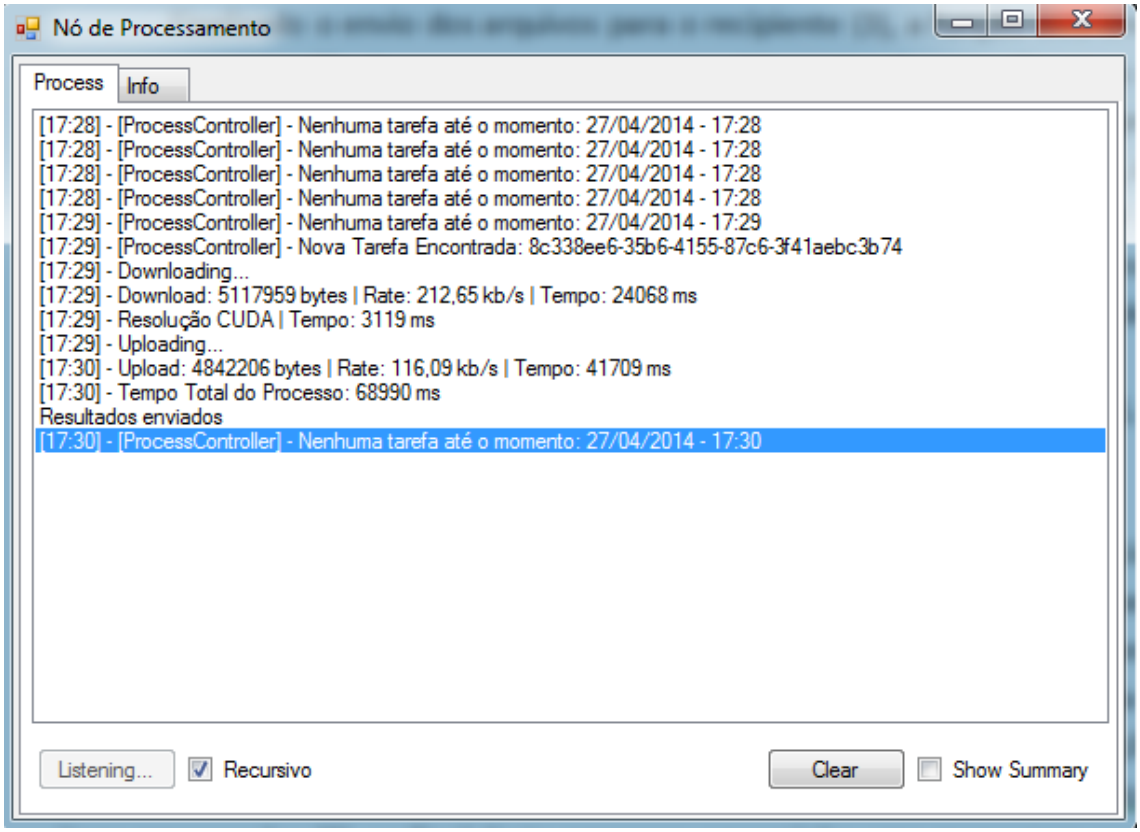

Figura 6 - Nó de processamento do sistema de decomposição LU

Ao finalizar o processo de decomposição LU, o sistema gera então novos arquivos de resultado e os armazena no recipiente do serviços de dados de itens processados (8),

A geração e envio dos arquivos de resultados segue o mesmo padrão que é utilizado ao enviar o arquivo para o recipiente de itens a processar, ou seja, formata e compacta cada linha da matriz em um único arquivo compondo vários arquivos compactados.

Os arquivos gerados são então gravados no formato blob no recipiente (8) com o seguinte padrão de nomenclatura:

“http://SUACONTA.blob.core.windows.net/RECIPIENTE/ID-ORDEM-NUMERO.zip”,

Onde SUACONTA representa o nome utilizado para cadastro nos serviços de armazenamento no Windows Azure, RECIPIENTE representa o nome do recipiente onde serão armazenados os arquivos da matriz LU resultante, e ID-ORDEM-NUMERO.zip representa o nome de cada arquivo submetido, onde ID é a identificação do projeto de execução cadastrado no banco de dados, ORDEM representa a ordem da matriz submetida e

NUMERO, representa a parte compactada do arquivo representando um numero de 0 a ORDEM-1 da matriz submetida

Ao enviar os arquivos de resultados, o sistema utiliza o modelo REST para encaminhá-los, nos mesmos moldes dos arquivos que foram enviados anteriormente com a matriz de elementos a ser decomposta, com a utilização de threads seguindo também o padrão produtor/consumidor.

Ao final deste processo, o nó de processamento encaminha uma mensagem com os dados e resumo dos tempos desta execução para a fila de itens processados (9), utilizando também o modelo REST sobre o protocolo HTTP, com o conteúdo da mensagem estruturados através de uma cadeia caracteres no formato JSON.

A função de trabalho por sua vez, monitorando a fila de itens processados, verifica a presença da mensagem, pois acessa frequentemente a fila a cada 1 segundo,visualiza e processa seu conteúdo (10), enviando seus resultados para o banco de dados (11), excluindo a mensagem da fila no final do processo, onde por fim ficará disponível para que o usuário tenha acesso ao histórico e ao arquivo de resultado final da decomposição LU (12) através de uma tabela de resultados.

A página da função web (Figura 7), disponibiliza o histórico dos dados gerados em um nó de processamento. Esta figura é representada essencialmente por uma tabela, que contém como colunas a data em que a execução do processamento foi feita, o tempo inicial de envio dos arquivos, a ordem utilizada na matriz, o tempo de *download* (cópia) dos arquivos do recipiente de itens a processar para o nó de processamento, o tempo de processamento na GPU com a utilização de CUDA, o tempo de *upload* (cópia) dos arquivos do nó de processamento para o recipiente de itens processados, o tempo total utilizado no processo e, para cada linha desta tabela, possui uma subtabela que mostra os resultados das transferências de arquivos utilizando a internet. Estes dados, serão utilizados para mensurar diferentes cenários, utilizando diferentes tipos e velocidade de conexão com a internet. Nesta tabela são exibidos apenas os resultados que obtiveram sucesso em todos os passos elencados.

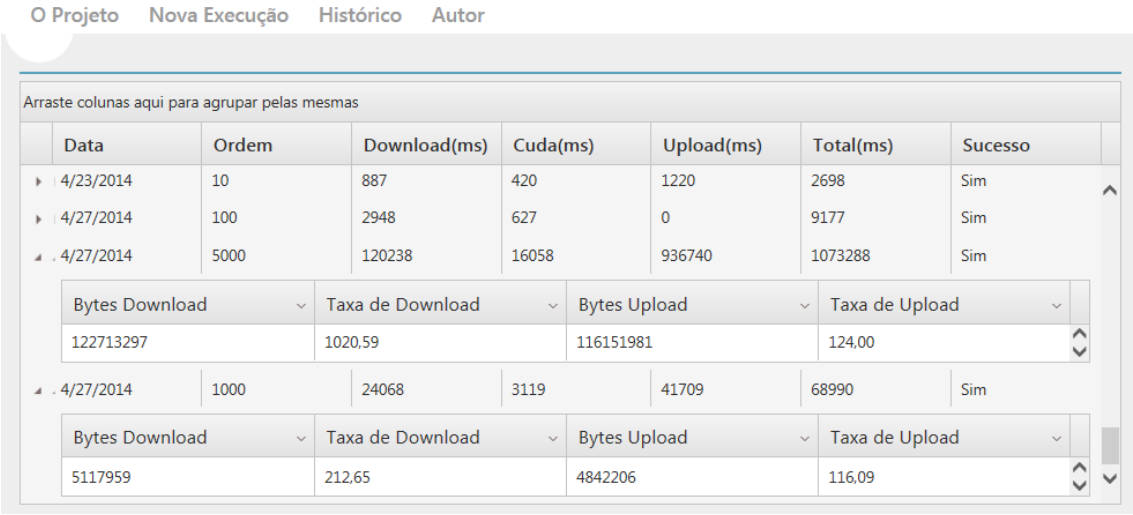

Figura 7 - Página de histórico de processamento da função web

6. RESULTADOS

6.1 Decomposição LU

Para estabelecer os critérios comparativos e medir o *speedup* estendido alcançado pelo sistema CloudLU, estudamos três diferentes algoritmos de decomposição LU (APÊNDICE C). O primeiro, que serve de base lógica para os demais, é a decomposição LU executada sequencialmente. O segundo faz uso de *threads* com auxilio da plataforma .NET e o terceiro utiliza o modelo CUDA baseado na linguagem C++. Este último, com chamadas ao código através da linguagem C# com o auxilio do projeto managedCUDA (APÊNDICE B).

Foram criadas matrizes de teste nas ordens de: 10, 100, 1000, 5000 e 10000 elementos, estabelecendo o critério inicial de testes local. Essas matrizes foram utilizadas para testar e mensurar o tempo de execução da decomposição LU em cada algoritmo.

Para cada teste foi medido o tempo de criação do arquivo, com seus respectivos tamanhos em bytes para as matrizes de testes, o tempo de leitura deste mesmo arquivo, o tempo de processamento do algoritmo de decomposição LU, o tempo de gravação do resultado em um arquivo local e o tempo total gasto no processo. Este último tempo, foi utilizado para quantificar o *speedup* estendido entre os algoritmos e a taxa de transferência média utilizada nas movimentações de dados para criar, ler e salvar os dados, através do uso do disco rígido primário evidenciado para o equipamento de teste.

O tamanho em bytes definido para cada matriz de testes, que compõem as tabelas de resultados, é baseado na quantidade de caracteres de cada número, de acordo com a definição do arquivo de testes definido no APÊNDICE F.

A taxa de transferência média foi calculada em megabits por segundo, sendo este parâmetro calculado baseado no número de bytes de cada arquivo de testes convertidos para megabit dividido pela média entre os tempos de movimentação de dados, convertido para segundo. Esta taxa de transferência foi utilizada para comparar as diferenças entre os algoritmos no sistema distribuído com o *speedup* estendido.

Algoritmo de decomposição LU sequencial

O algoritmo foi implementado na linguagem Visual C# e foi realizado execuções (Tabela 1) de acordo com as ordens de testes evidenciadas.

		Tempo em milissegundos					
Ordem	Bytes	Criar	Ler	Processar	Salvar	Total	Mbps
10	1000	7	6	8	9	30	1,04
100	100000	44	12	10	43	109	23,12
1000	10000000	2841	903	10318	2748	16810	35,26
5000	250000000	69071	22742	1261347	65434	1418594	36,39
10000	1000000000	277525	90402	10099239	271458	10738624	35,80

Tabela 1 - Medição de tempos para algoritmo de decomposição LU sequencial local

Ao concluir os testes, os resultados (Tabela 1) apresentaram o tempo total gasto no processo proporcional à ordem das matrizes testadas, juntamente com a taxa de transferência média atingida. O tempo total desta medição serviu como base para o cálculo do *speedup* estendido, sendo a partir deste, medidos os ganhos obtidos por outros algoritmos. É possivel notar que através destes resultados o tempo total aumenta conforme a ordem da matriz aumenta. Porém, a taxa de transferência permanece relativamente estável.

Ao efetuar a execução do algoritmo sequencial, pode-se verificar o uso da CPU (Figura 8), através do aplicativo de gerenciamento de memória presente nos computadores que utilizam o sistema operacional Microsoft Windows. Durante o processamento da matriz de ordem 10.000, pode ser observado através do segundo quadro - que representa um dos núcleos de processamento do equipamento utilizado - o momento em que um bloco de dados é processado. Através do mesmo quadro, pode-se verificar a queda do processamento, representando a finalização do processamento de um dos blocos da matriz e seguido da retomada do processamento, quando se inicia o processamento do bloco seguinte. Entre o processamento de um bloco e outro pode-se verificar o incremento de processamento em outro núcleo, evidenciado pelo processamento do pivotamento no primeiro quadro. Pode-se também observar que cerca de apenas ¼ do poder de processamento do equipamento é utilizado, isto representa 1 dos núcleos do processamento, comprovando visualmente, a execução sequencial do algoritmo sem uso de paralelismo.

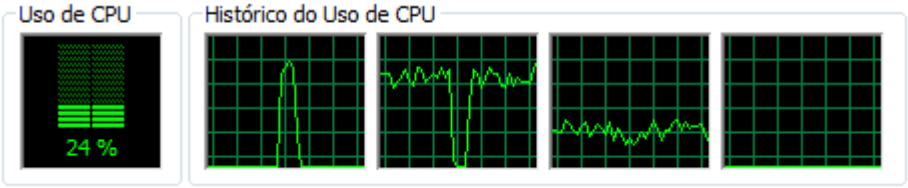

Figura 8 - Histórico de uso de CPU executando um algoritmo sequencial

Algoritmo de decomposição LU com a utilização de threads

Com o intuito de estabelecer uma medição intermediária entre o algoritmo de decomposição LU sequencial e o algoritmo que utiliza CUDA, foi criado um algoritmo que utiliza a biblioteca Parallel da plataforma .NET. Esta biblioteca, de acordo com (ALBAHARI, 2014), "fornece implementação paralelas de cada um dos operadores de consulta padrão. Essencialmente, eles funcionam dividindo a sequência de entrada em pedaços que são executados em threads diferentes, retornando os resultados de volta em uma única sequência de saída".

A implementação é semelhante à utilização do "*for*" quando se pretendem utilizar laços em linguagens como a própria C#, C e C++ por exemplo. Porém ao se estabelecer um laço se utiliza "Parallel.for(...)", ao invés de apenas "for(...)".

Os resultados de testes (Tabela 2) foram então mensurados para o algoritmo com uso de threads através da biblioteca Parallel.

		Tempo em milissegundos					
Ordem	Bytes	Criar	Ler	Processar	Salvar	Total	Mbps
10	1000	7	1	66	9	83	1,35
100	100000	44	13	107	42	206	23,12
1000	10000000	2841	959	3116	2695	9611	35,24
5000	250000000	69071	23080	344679	64589	501419	36,51
10000	1000000000	277525	90248	2775306	269433	3412512	35,92

Tabela 2 - Medição de tempos para algoritmo de decomposição LU com uso de threads

Ao concluir os testes, é possível perceber que os resultados (Tabela 2) apresentados possuem semelhanças em relação ao algoritmo sequencial, ou seja, os tempos aumentam conforme a ordem da matriz. Porém os tempos medidos são menores em relação à execução sequencial apenas a partir da matriz de ordem 1000.

No o uso da CPU (Figura 9), através do aplicativo de gerenciamento de memória, é possível notar que com a utilização de threads neste caso, com a execução da matriz de ordem 10.000, ocupa todo o poder de processamento dos quatro núcleos presentes no equipamento, diferente do mostrado anteriormente (Figura 8) da execução sequencial, esta abordagem é mais eficiente em tempo de processamento, porém não deixa recursos de processamento disponíveis.

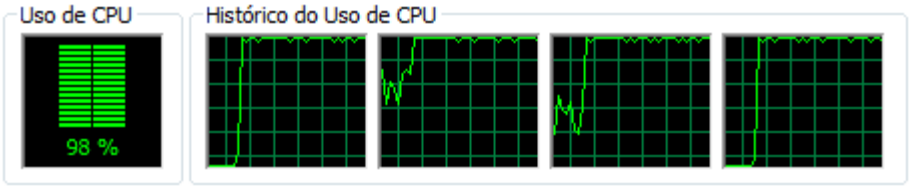

Figura 9 - Histórico de uso de CPU executando um algoritmo com a utilização de threads

Algoritmo de decomposição LU com a utilização de GPU e CUDA

O algoritmo foi implementado na linguagem C++ por (OVESEN, 2014) de acordo com o modelo CUDA.

Para incorporar o algoritmo ao sistema CloudLU estudado, foi utilizado o projeto managedCuda, que promoveu a interoperação entre as linguagens C# e C++, permitindo o acesso do sistema proposto ao algoritmo implementado, manipulando o algoritmo e recebendo os resultados da decomposição LU, para posterior medição dos tempos.

Para que a interoperabilidade entre as diferentes linguagens de desenvolvimento fosse possível, uma parte do algoritmo foi adaptado à linguagem Visual C#, ficando apenas o algoritmo que é efetivamente executado na GPU, inalterado.

Para as ordens da matriz de testes foram mensurados os tempos (Tabela 3) para o algoritmo com uso de GPU através do modelo CUDA.

CUDA		Tempo em milissegundos					
Ordem	Bytes	Criar	Ler	Processar	Salvar	Total	Mbps
10	1000	7	4	59	9	79	1,14
100	100000	44	11	103	44	202	23,12
1000	10000000	2841	934	1210	2801	7786	34,81
5000	250000000	69071	22869	10906	64541	167387	36,57
10000	1000000000	277525	89819	78417	270445	716206	35,89

Tabela 3 - Medição de tempos para algoritmo de decomposição LU com uso de GPU local

Ao concluir os testes utilizando o modelo CUDA, os resultados (Tabela 3) apresentaram também um aumento dos tempos conforme a ordem das matrizes. Neste caso, porém, os tempos são melhores que a execução com uso de *threads* a partir da ordem 100 (Tabela 2) e maiores em relação a execução sequencial a partir da ordem 1000 (Tabela 1).

É possivel perceber o uso do processador, através do histórico de uso de CPU (Figura 10), durante a execução do algoritmo de decomposição LU, utilizando GPU. O que são observados através do gráfico de uso apresentado na imagem, são as movimentações de dados entre a memória principal e a memória da GPU e o gerenciamento do código, que necessita o uso da CPU.

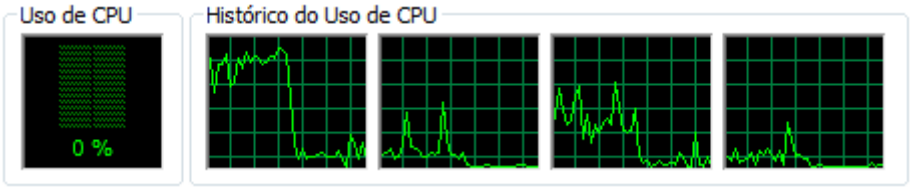

Figura 10 - Histórico de uso de CPU executando um algoritmo com a utilização de CUDA

As medições dos tempos de processamento quando feito através de GPU, leva em consideração o tempo necessário à cópia dos dados da matriz primária da memória principal para a memória da GPU, e o seu respectivo resultado após o final do processamento para a memória principal.

Ao final das medições dos tempos de processamento dos diferente algoritmos, executados localmente, as medidas de tempo total de cada algoritmo com as respectivas ordens da matriz fornecem os tempos necessários ao calculo do *speedup* estendido (Tabela 4).

	Tempo em milissegundos		
Ordem	Sequencial	Paralelo	Cuda
10	30	83	79
100	109	206	202
1000	16810	9611	7786
5000	1418594	501419	167387
10000	10738624	3412512	716206

Tabela 4 - Comparativo entre os diferentes tempos mensurados pelos diferentes algoritmos de decomposição LU

Podemos notar que o algoritmo de decomposição LU que utiliza CUDA, passa a ser mais vantajoso computacionalmente a partir da execução de uma matriz de ordem 1000 (Tabela 4). Isso se deve principalmente ao fato de que ao se movimentar dados da memória principal para a memória da GPU ocasiona perda de rendimento e conseqüentemente aumento de tempo.

> Seja um algoritmo paralelo que usa p processadores (p pode depender de n) e termina no tempo Tp(n) e seja T∗(n) o tempo requerido pelo algoritmo sequencial ótimo para resolver o mesmo problema, define-se como *speedup* a medida relativa entre esses dois tempos. Em outras palavras, *speedup* é a medida, dada pela equação 2, que indica o número de vezes que o programa paralelo é mais rápido que o sequencial. (MILANI, 2008) e (BERTSEKAS e TSITSIKLIS, 1997).

$$\mathrm{Sp(n)} = \frac{T*(n)}{Tp(n)} \tag{2}$$

Comparamos agora os tempos totais obtidos através da divisão do tempo mensurado pelo algoritmo sequencial pelo algoritmo que utiliza CUDA o *speedup* estendido, calculado em relação a cada um dos algoritmos e as ordens de suas matrizes (Figura 11).

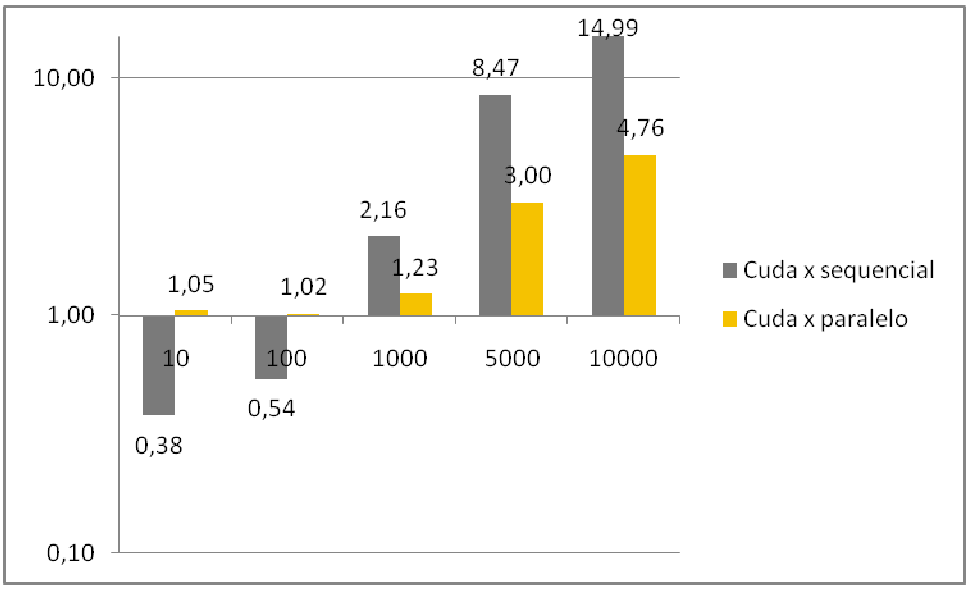

Figura 11 - Speedup em relação ao algoritmo em Cuda e aos algoritmos sequencial e paralelo

É importante destacar o valor para uma matriz de ordem 10000 (Figura 11). Nesse caso, o algoritmo que utiliza CUDA é aproximadamente 15 vezes mais rápido que o sequencial, e 5 vezes mais rápido que o paralelo baseado em *threads*. Esta mesma métrica de comparação será utilizada ao comparar os resultados do sistema CloudLU com a execução do algoritmo sequencial e paralelo para então mensurar a viabilidade deste sistema, que será apresentado na seção seguinte.

6.2 Sistema CloudLU - processamento usando a nuvem

Baseado no sistema CloudLU desenvolvido realizamos os mesmos testes com as ordens de matriz de decomposição LU: 10, 100, 1000, 5000 e 10000, seguindo o fluxo da aplicação mostrado anteriormente. Deve-se levar em consideração que os resultados destes testes foram mensurados utilizando o equipamento e conexão descritos anteriormente. Veremos a seguir a influência da conexão utilizada na eficiência do sistema CloudLU.

Para cada uma das ordens definidas como parâmetros de comparação de testes foi medido:

- o tempo que é necessário para efetuar a cópia inicial do arquivo que contém os dados da matriz de elementos para o sistema na nuvem, que pode ser comparado com o tempo de criação do arquivo no teste sequencial;

- o tempo necessário para fazer o download destes arquivos para o nó de processamento, que pode ser comparado ao tempo de leitura do arquivo no teste sequencial;

- o tempo de processamento utilizando CUDA, que pode ser comparado ao tempo de execução do algoritmo sequencial;

- o tempo de cópia destes dados já processados para o recipiente de resultados na nuvem, que pode ser comparado ao tempo de criação do arquivo de resultados na versão sequencial dos testes.

- o tempo total transcorrido durante todo o processo. Este tempo leva em consideração a soma dos tempos anteriores, assim como o tempo entre um processo e outro, extraídos do sistema de gerenciamento (Tabela 5).

	Média de tempos em milissegundos				
	Upload			Upload	
Ordem	Inicial	Download	Cuda	Resultado	Total
10	1643	216	132	1113	3356
100	4325	1479	130	3579	9763
1000	24992	7337	2937	30010	65514
5000	251299	149559	17997	240027	659118
10000	586002	295570	40523	595224	1517708

Tabela 5 - Médias de tempos das execuções no sistema CloudLU

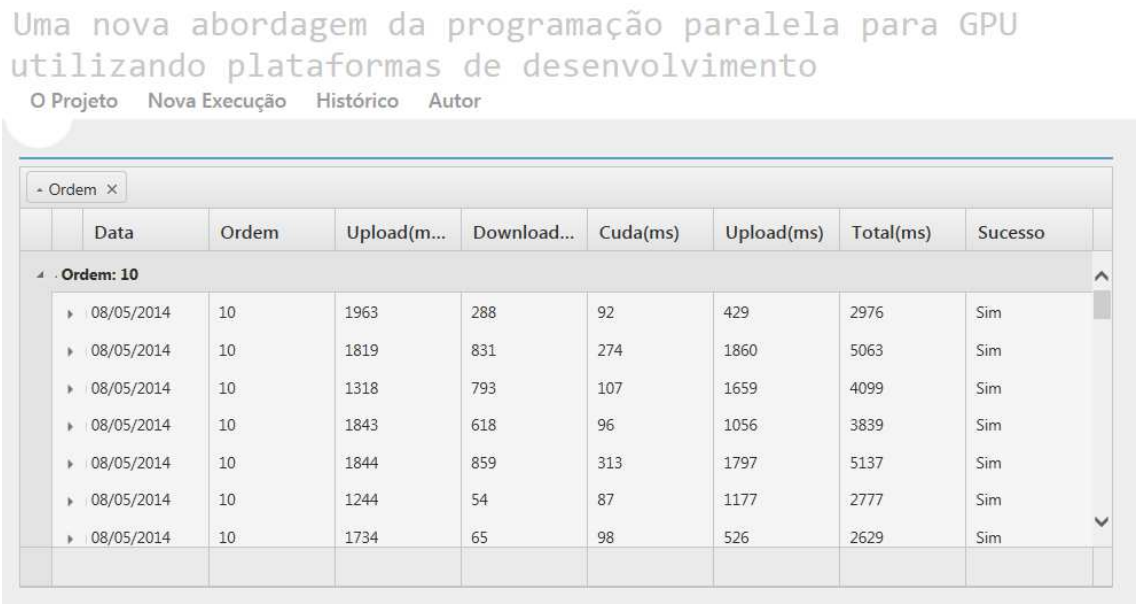

Figura 12 - Tabela do sistema gerenciamento mostrando as médias entre as execuções agrupadas pela ordem da matriz.

Como é possível notar nos dados apresentados, os tempos relativos ao tráfego de dados (Tabela 5) entre os sistemas constituem um gargalo para esta abordagem, principalmente para a conexão utilizada com a Internet. Estes números podem obter diferenças significativas se comparados os testes feitos, utilizando uma conexão de rede local, que não é o caso desta abordagem.

Os resultados apresentam uma pequena diferença em relação à movimentação de dados. Isso ocorre porque o sistema CloudLU utiliza a internet como meio de comunicação e a velocidade de conexão apresenta variações ao longo do processo que não ocorrem no caso do algoritmo executado localmente.

Levando em consideração apenas os valores totais de tempo de cada uma das execuções, os valores relativos a execução do sistema CloudLU obter os dados são apresentados na Tabela 6.

	Tempo em milissegundos		
Ordem	Sequencial	Paralelo	CloudLU
10	30	83	3356
100	109	206	9763
1000	16810	9611	65514
5000	1418594	501419	659118
10000	10738624	3412512	1517708

Tabela 6 - Valores totais mensurados pela versão sequencial, a versão paralela em comparação aos dados mensurados com o sistema CloudLU

Podemos verificar que, ainda que se utilize uma conexão com a internet, o processamento em CUDA juntamente com o sistema CloudLU é 7,08 vezes mais rápido que a versão sequencial e 2,25 vezes mais rápido em relação à versão paralela dos algoritmos de decomposição LU para n = 10000 (Figura 13).

Para obter os valores apresentados na Figura 13, o sistema CloudLU executou 10 vezes o algoritmo para cada ordem apresentada, obtendo os valores do speedup e também o seu respectivo desvio-padrão.

Se compararamos ainda o valor do *speedup* estendido, relativo à n = 5.000, verificamos que o sistema CloudLU é 2,15 vezes mais rápido em relação ao sequencial.

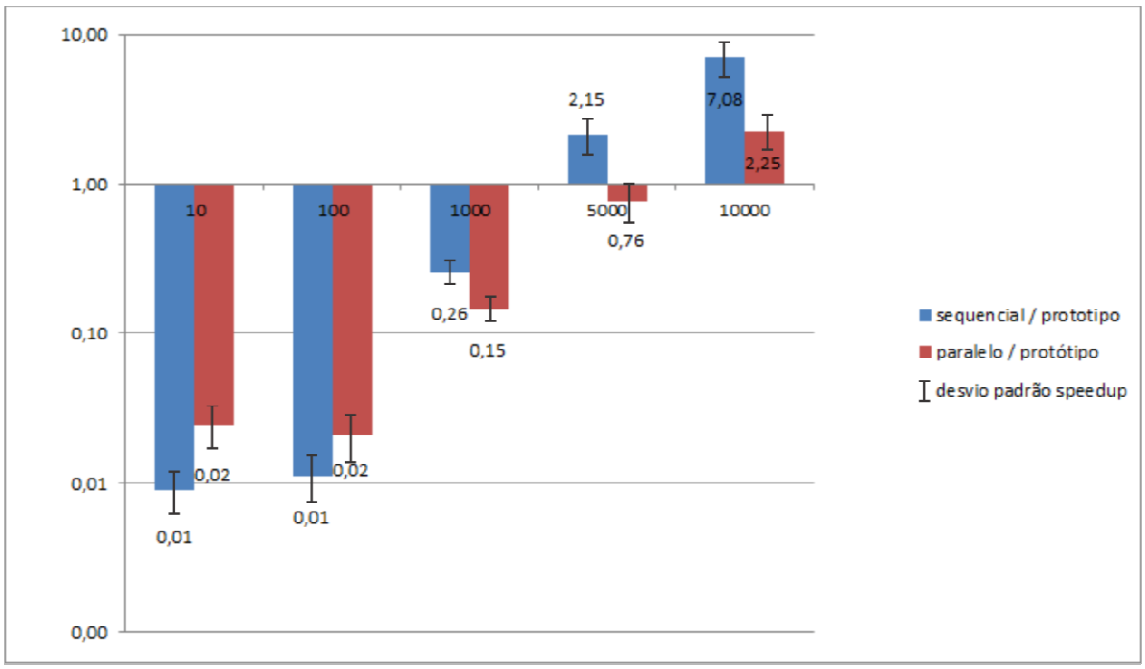

Figura 13 - Speedup alcançado relacionando o algoritmo sequencial e paralelo ao sistema CloudLU

Para uma conexão com internet mais lenta (velocidade de *download* de 10Mbps e de *upload* de 1 Mbps), o processamento de matriz n= 10000 foi de apenas 1,45, comprovando que a internet nesta abordagem constitui claramente um gargalo.

Portanto, considerando este gargalo que a conexão com a internet utilizada constitui para a abordagem apresentada do sistema CloudLU, este sistema é viável apenas quando se trafega grande quantidade de dados. No caso da decomposição LU, isso corresponde a uma matriz de ordem 5000.

Ainda considerando o fator conexão com a internet, é necessário evidenciar outros dados que foram mensurados pelo sistema CloudLU, como a taxa de transferência média, alcançada em cada um dos testes medidos.

Para mensurar a taxa de transferência foi calculada a quantidade de bytes totais transferidos em cada uma das etapas do processo, sendo eles: a cópia do arquivo inicial da matriz de elementos para o sistema em nuvem; o *download* do sistema em nuvem para o nó de resultados que constitui a mesma quantidade de bytes da cópia inicial; e posteriormente a cópia dos resultados de volta para o sistema em nuvem. Com isto, utilizando a quantidade de bytes trafegados juntamente com os tempos necessários a este processo elencados na Tabela 5, é possível determinar a taxa de transferência média, em Mbps (mega bits por segundo) para cada uma das ordens testadas.

É importante lembrar que os dados trafegados a partir do momento em que se faz o *upload* inicial dos arquivos são compactados. A descompactação dos dados é feita ao realizar o *download* destes dados no nó de processamento e, após a execução da decomposição LU, estes dados são novamente compactados para reduzir o tempo e a massa de dados trafegados, constituindo também uma diferença em relação aos dados obtidos na execução local.

Inicialmente foi feita a conversão dos bytes multiplicando por 8, representando assim o mesmo número em bits; o tempo levado em cada uma das etapas foi então convertido de milissegundos dividindo seus valores por 1000, representando assim o mesmo número em segundos, constituindo a taxa de transferência em bits por segundo. Para representar os valores em mega bits por segundo, foi feita a divisão deste último número por 1048576 (o mesmo que multiplicando por 1024 para converter em kilobits por segundo, e em seguida multiplicados por 1024, convertendo então para mega bits por segundo) constituindo os valores das respectivas taxa de transferência (Tabela 7) e consequentemente a taxa de transferência média.

Sendo B, o numero em bytes transferido em cada uma das etapas, e t o tempo em milissegundos demandado por cada uma das etapas, a equação (3), fornece a taxa de transferência em Mbps (mega bits por segundo):

$$Mbps = \frac{\frac{B*8}{t/1000}}{1048576} \quad (3)$$

	Upload inicial			Download			Upload dos resultados		
Ordem	tempo (ms)	Bytes	Mbps	tempo (ms)	Bytes	Mbps	tempo (ms)	Bytes	Mbps
10	1643	1801	0,01	216	1801	0,06	1113	1817	0,01
100	4325	56626	0,10	1479	56626	0,29	3579	57927	0,12
1000	24992	4560861	1,39	7337	4560861	4,74	30010	4644802	1,18
5000	251299	109557465	3,33	149559	109557465	5,59	240027	111343065	3,54
10000	586002	429298954	5,59	295570	429298954	11,08	595224	435914485	5,59

Tabela 7 - Taxa de transferência apurada em mega bit por segundo nas movimentações de dados no sistema CloudLU

Considerando a taxa de transferência alcançada em cada um dos testes realizados, a média entre as conexões de *upload*, com a utilização da conexão de internet descrita (*download* de 26,52 mega bits por segundo e *upload* de 20,83 megabits por segundo), é possível constatar que a taxa de transferência não atinge o máximo da velocidade disponível (Tabela 8).

Ordem	Mbps upload	% de uso	Mbps download	%
10	0,01	0,05	0,06	0,31
100	0,11	0,54	0,29	1,40
1000	1,29	6,18	4,74	22,77
5000	3,43	16,48	5,59	26,83
10000	5,59	26,83	11,08	53,20

Tabela 8 - - Porcentagem de utilização da conexão com a internet disponível

Ao se tratar de uma conexão com a internet compartilhada, nem sempre é possível utilizar toda a velocidade disponível, devido à distribuição que há entre os usuários desta rede.

Porém, se levarmos em consideração a possibilidade de que o sistema CloudLU possa ser utilizado em uma rede local, a qual utiliza uma rede de alta velocidade com a utilização de cabos, um novo cenário pode ser descrito.

Supondo a utilização de uma rede local gigabit, que corresponde a 1000 mega bits por segundo, a qual interconecta o nó de gerenciamento juntamente com o nó de processamento em uma nuvem privada, novas medições de tempo podem ser extraídas desta suposição. Levando em consideração o fator de utilização mensurado (Tabela 8), apresenta-se novas taxas de transferências (Tabela 9).

Ordem	Mbps upload	%	Mbps download	%
10	0,52	0,05	1,92	0,31
100	5,58	0,54	20,62	1,40
1000	64,33	6,18	237,56	22,77
5000	171,63	16,48	633,82	26,83
10000	279,42	26,83	1031,85	53,20

Tabela 9 - Taxa de transferência suposta em uma conexão de rede gigabit

Com isto é possível também mensurar novos tempos (Tabela 10) entre os uploads e downloads do sistema CloudLU, proporcionalmente relacionados à taxa de transferência suposta (Tabela 9).

	Tempos em milissegundos				
	Upload			Upload	
Ordem	Inicial	Download	Cuda	Resultado	Total
10	33	6	132	22	445
100	86	38	130	72	577
1000	500	191	2937	600	4465
5000	5026	3889	17997	4801	31949
10000	11720	7685	40523	11904	72222

Tabela 10 - Tempos de processamento supondo a utilização de uma rede gigabit

Supondo os novos tempos de processamento e movimentação de dados, baseado em uma conexão gigabit (Tabela 11), novos valores de *speedup* estendido, puderam ser obtidos (Figura 14).

TOTAL	Tempo em milissegundos		
	Sequencial	Paralelo	CloudLU
10	30	83	445
100	109	206	577
1000	16810	9611	4465
5000	1418594	501419	31949
10000	10738624	3412512	72222

Tabela 11 - Valores totais mensurados pela versão sequencial, paralela e a do sistema CloudLU supondo a utilização de uma rede gigabit

É possível demonstrar (Figura 14) que ao utilizar uma rede gigabit o sistema CloudLU atinge valores até 149 vezes mais rápido que a versão do algoritmo sequencial de decomposição LU e, até 47 vezes mais rápido que a versão paralela do mesmo algoritmo ao processar uma matriz de ordem 10.000 utilizando um sistema distribuído em nuvem com a utilização de uma rede local gigabit.

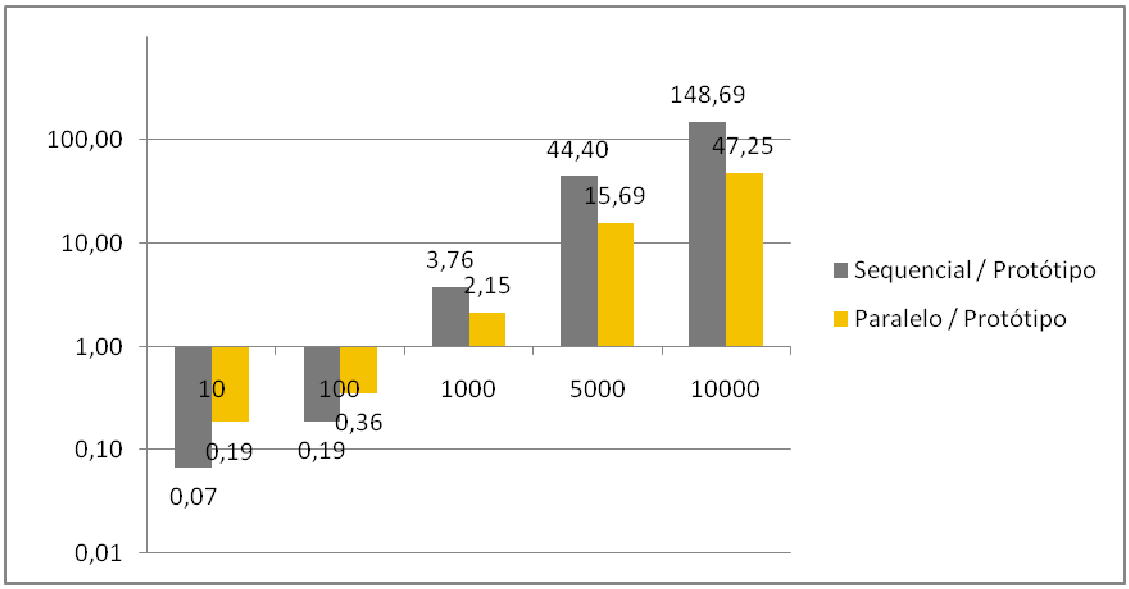

Figura 14 - Speedup alcançado relacionando o algoritmo sequencial e paralelo ao sistema CloudLU supondo a utilização de uma rede gigabit

Para melhor ilustrar os dados apresentados, o *speedup* alcançado em relação apenas com o algoritmo CUDA em relação ao algoritmo sequencial de decomposição LU em cada um dos casos mensurados anteriormente, juntamente com a nova proposição de uso com uma rede gigabit é apresentado na Figura 15.

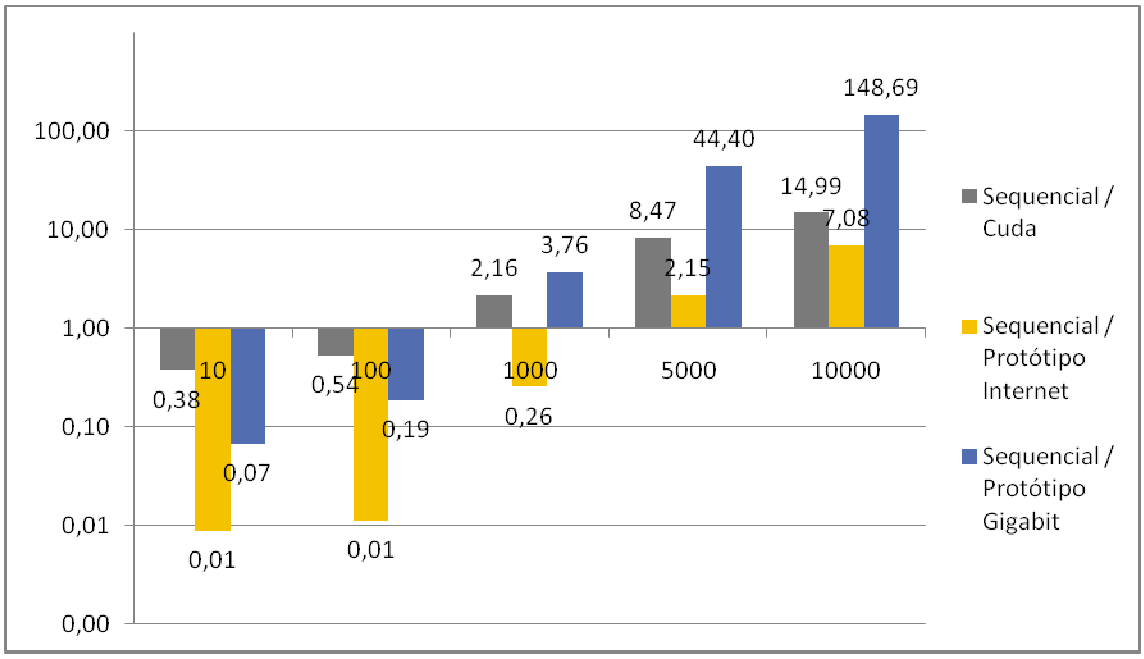

Figura 15 - Speedup alcançado ao relacionar os algoritmos CUDA e o sequencial utilizando diferentes cenários.

É possível perceber através dos dados apresentados pelo gráfico (Figura 15), que há diferenças entre os cenários propostos e consequentemente no *speedup* medido entre esses cenários.

A primeira diferença se dá em relação ao *speedup* da relação Sequencial/Cuda que foi medida através de testes efetuados localmente no equipamento descrito e as relações Sequencial/CloudLU Internet e Sequencial/CloudLU Gigabit, que foram ambas medidas utilizando o sistema CloudLU e diferentes tipos de conexões.

Na versão Sequencial/Cuda, as medições de tempo, foram feitas através da leitura e gravação sequencial de arquivos localmente, sem qualquer tipo de compactação e/ou processamento, ficando seus tempos medidos sujeitos a este cenário (Tabelas 1, 2 e 3).

Em contrapartida, nas versões Sequencial/CloudLU Internet e Sequencial/CloudLU Gigabit, os arquivos trafegados entre os nós, ficaram sujeitos à compactação dos dados e a utilização de threads para a transmissão e recepção, como abordado anteriormente, sendo então processados paralelamente, alcançando medições diferentes das suas respectivas associações às versões sequenciais destes cenários, conforme evidenciado pelas Tabelas 7 e 9.

Outra métrica que pode ser utilizada para verificar a diferença encontrada entre o *speedup* estendido das diferentes abordagens é a taxa de transferência relativa a cada um dos cenários, observando a ordem da matriz de 10.000 elementos é possível notar que a taxa de

transferência média dos dados atingiu 35,80 Mbps (Tabela 1), enquanto que na versão do sistema CloudLU, considerando a média entre dois uploads e um download, atingiu uma média de 7,2 Mbps (Tabela 7) e na suposição feita, utilizando uma rede gigabit, a média para a mesma matriz foi de 530,23 Mbps (Tabela 9), também para dois uploads e um download. A taxa de transferência, portanto, está diretamente relacionada ao calculo do *speedup* estendido, quando se inclui a movimentação de dados e seus respectivos tempos, ao se comparar diferentes tipos de sistemas, como os elencados nesta proposição.

Portanto, apesar de serem diferentes cenários para o mesmo estudo de caso, o sistema CloudLU apresentou resultados satisfatórios, ainda que nos resultados apresentados pelo cenário Sequencial/CloudLU com a utilização de uma conexão gigabit seja um cenário calculado e não testado efetivamente.

Com os resultados apresentados, é possível concluir que o sistema CloudLU possui resultados satisfatórios apenas com a utilização de matrizes com ordens à partir de 5.000, onde os ganhos obtidos são evidentemente notados.

7. CONCLUSÕES E CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentada a relação entre um sistema distribuído com utilização de nuvem, em especifico o Microsoft Azure e o processamento através do modelo CUDA com utilização de GPU. Dentro destes conceitos, foram revisados os tópicos relacionados a cada um deles, utilizando um estudo de caso com a decomposição LU através de um algoritmo sequencial, um algoritmo paralelo com o uso de *threads* e outro algoritmo paralelo com uso de GPU. Um sistema CloudLU foi desenvolvido para demonstrar o funcionamento em conjunto destes conceitos e foram obtidas métricas de comparação utilizando o *speedup* estendido, obtendo resultado 7 vezes mais rápido, ao relacionar o processamento CUDA com a versão sequencial apresentada utilizando uma conexão com a internet e, 148 vezes mais rápido, ao supor a utilização uma conexão gigabit local.

O sistema CloudLU aqui desenvolvido buscou demonstrar que, a utilização do processamento através de GPU aliado ao conceito de um sistema distribuído como a nuvem, oferece grandes possibilidades e maior visibilidade para os produtos que o compõem.

Ainda no conceito de sistemas distribuídos, a empresa Nvidia possui a tecnologia Nvidia Grid, que permite a transferência do processamento da CPU para a GPU utilizando determinadas tecnologias de desenho e engenharia disponíveis no mercado, facilitando o processamento em computadores remotos em ambientes virtualizados. Isto demonstra a crescente importância que se tem nestes ambientes virtuais e distribuídos, os quais necessitem de produtividade oferecida através do paralelismo presente nas GPUs.

Embora o sistema CloudLU aqui apresentado tenha tido inicialmente o intuito de demonstrar a junção dos conceitos de GPU e nuvem, foi identificada uma possível aplicação exclusiva para sistemas distribuídos. Se trata de adaptar este sistema para efetuar testes de carga e medir a eficiência dos recursos de uma nuvem. Isto é possível a partir do momento em que se isole o processamento destinado à GPU e o direcione para processamento nos nós que compõem uma determinada nuvem, estimando os tempos de processamento, os tempos de transferência de dados e, inclusive, a persistência em arquivo ou banco de dados.

Dependendo da maneira como for feita esta adaptação, a própria decomposição LU pode ser utilizada para testar uma nuvem, assim como já é feito com o uso do LINPACK para testar os supercomputadores.

Uma iniciativa parecida foi tomada na criação do CLOUD500, com o objetivo de medir e pontuar a eficiência de nuvens, baseando-se na medição de seus recursos em relação

aos seus respectivos preços. Com isso obtém-se uma referência para classificar as nuvens e suas respectivas empresas em uma lista comparativa.

Ainda que se utilize na pontuação do CLOUD500 diferentes ferramentas para medir diferentes recursos, não encontramos uma única ferramenta que englobe os testes de todos os recursos oriundos das nuvens submetidas para classificação, e também uma automatização destes testes. O sistema CloudLU proposto pode ser uma dessas ferramentas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBAHARI, J. *Threading in C#*. Acesso em 20 de Abril de 2014, disponível em http://www.albahari.com/threading/part5.aspx

AMDAHL, G. *"Validity of the single processor approach to achieving large scale computing capabilities"*.(PDF) AFIPS ConferenceProceedings (30): 483-485, Acesso em 20 de Abril de 2014, disponível em http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf

American National Standards Institute / Institute of Electrical and Electronics. Engineers: *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985, New York, 1985.

BERTSEKAS, D. P.; TSITSIKLIS, J. N..*Parallel and Distributed Computation: NumericalMethods*. Athena Scientific, 1997.

CHEESO; JAANS. *DotNetZip Project*. 2007. Acesso em 20 de Abril de 2014, disponível em http://dotnetzip.codeplex.com/

CLOUD500. *Ranking the Top 500 Iaas Cloud Providers*. Acesso em 19 de Maio de 2014, disponível em http://cloud500.org/index.php

CROCKFORD, D. RFC4627, *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006. Acesso em 20 de Abril de 2014, disponível em: http://tools.ietf.org/html/rfc4627

DONGARRA, J.; LUSZCZEK, P.; PETIT, A.; *The LINPACK BENCHMARK: past, present and future*, 2003. Acesso em 21 de Abril de 2014, disponível em http://www.netlib.org/utk/people/JackDongarra/PAPERS/146_2003_the-linpack-benchmark-past-present-and-future.pdf

DPALLMANN. *Azure Storage Sample*. 2011. Acesso em 20 de Abril de 2014, disponível em http://azurestoragesamples.codeplex.com

European Computer Manufacturers Association, *ECMAScript Language Specification 3rd Edition*, December 1999, Acesso em 20 de Abril de 2014, disponível em http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf

FIELDING, R. T. *Representational State Transfer*, 2000. Acesso em 20 de Abril de 2014, disponível em http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

GPGPU.ORG, *General Porpouse Graphics Process Unit*. Acesso em 20 de Abril de 2014, disponível em http://gpgpu.org/about

HALFHILL, T. R. *Looking Beyond Graphics. Fermi Architecture for High-Performance Computing* | NVIDIA, 2009. Acesso em 20 de Abril de 2014, disponível em http://www.nvidia.com/content/PDF/fermi_white_papers/T.Halfhill_Looking_Beyond_Graphics.pdf

KUNZMI. *ManagedCuda Project*. 2011Disponivel em http://managedcuda.codeplex.com

LUEBKE, D.; HUMPHREYS, G. *How Things Works*, 2007. Acesso em 20 de Abril de 2014, disponível em http://www.cs.virginia.edu/~gfx/papers/pdfs/59_HowThingsWork.pdf

MICROSOFT. *Introdução a consultas LINQ (C#).* Acesso em 20 de Abril de 2014, disponível em http://msdn.microsoft.com/pt-br/library/bb397906.aspx

MICROSOFT. *Introdução a Plataforma .NET*. Acesso em 20 de Abril de 2014, disponível em http://msdn.microsoft.com/pt-br/aa702903.aspx

MICROSOFT. *O ADO.NET Entity Framework*. Acesso em 20 de Abril de 2014, disponível em: http://msdn.microsoft.com/pt-br/data/aa937723.aspx

MICROSOFT. *Recursos do Visual C#.* Acesso em 20 de Abril de 2014, disponível em: http://msdn.microsoft.com/pt-br/vstudio/hh341490

MICROSOFT. *Visual Studio*. Acesso em 20 de Abril de 2014, disponível em http://www.visualstudio.com/pt-br/visual-studio-homepage-vs.aspx

MICROSOFT. *What is Azure?*. Acesso em 20 de Abril de 2014, disponível em http://msdn.microsoft.com/en-us/library/azure/ux/develop/dd163896.aspx

MICROSOTF. *Serviços de Nuvem*. Acesso em 20 de Abril de 2014, disponível em http://msdn.microsoft.com/pt-br/library/azure/jj155995.aspx

MILANI, C.R. *Estudo sobre a aplicação da Computação Paralela na resolução de sistemas lineares*. Pontifícia Universidade Católica do Rio Grande do Sul, 2008. 26p. Acesso em 20 de Abril de 2014, disponível em http://www.inf.pucrs.br/gmap/pdfs/Cleber/IP1%20-%20Cleber.pdf

NVIDIA, *Cuda-C Programming Guide*. Acesso em 20 de Abril de 2014, disponível em http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction

NVIDIA, *O que é computação com Gpu*, Acesso em 29 de Setembro de 2013, disponível em http://www.nvidia.com.br/object/what-is-gpu-computing-br.html

NVIDIA, *What is Gpu Computing*. Acesso em 20 de Abril de 2014, disponível em http://www.nvidia.com/object/what-is-gpu-computing.html#sthash.ZopzFzkL.dpuf

OVESEN, M. *Cuda Lu Decomposition*. Acesso em 20 de Abril de 2014, disponível em https://bitbucket.org/ovesen/cuda-lu-decomposition

PATTERSON, D. A. e J. L. HENNESSY (1998*). Computer Organization and Design*, Second Edition, Editora Morgan Kaufmann, p. 635.

TOP500 Supercomputing sites, 2014. Acesso em 20 de Abril de 2014, disponível em http://www.top500.org

TULLOCH, M. *Introducing Windows Azure for IT Professional*. Microsoft Press, 2013. Acesso em 20 de abril de 2014, disponível em http://blogs.msdn.com/b/microsoft_press/archive/2013/10/01/free-ebook-introducing-windows-azure-for-it-professionals.aspx

WEBBER, J.; Parastatidis, S.; Robinson, I**.** *Rest in Practices*, 1.ed : O'Reilly Media, 2010. 12p

WILT, N. *The CUDA Handbook*, A Comprehensive Guide to GPU Programming, Pearson Inc. 2013.viii, 231p.

APÊNDICE A

SERVIÇOS DE COMPUTAÇÃO

No núcleo da plataforma Windows Azure está a capacidade de executar aplicativos na nuvem. Windows Azure fornece quatro modelos diferentes: sites da web, máquinas virtuais, serviços em nuvem e serviços móveis. Estas quatro abordagens compreendem a parte de serviços de computação da plataforma Windows Azure, e elas podem ser usadas separadamente ou combinadas entre si para criar soluções mais complexas. (TULLOCH, 2013)

Dentro desta gama que compreende os serviços de computação, neste projeto utilizaremos apenas os serviços de nuvem pois, de acordo com (TULLOCH, 2013), permite criar, implantar e gerenciar múltiplas camadas de aplicativos na nuvem, podendo definir diversas funções para o aplicativo e distribuir processamento, além de ter a capacidade de utilizar as linguagens populares para estes aplicativos.

Em outras palavras, os serviços de computação do Windows Azure hospedam o sistema CloudLU que gerencia e disponibiliza os recursos necessários para a viabilidade deste, compreendendo duas funções básicas:

- Função Web – Uma função personalizada para programação de aplicativo Web com suporte pelo IIS 7 e pelo ASP.NET. A vantagem de usar este tipo de função é que a configuração do IIS é feita para você. Esta função é melhor usada para fornecer um *frontend* baseado na Web para seu serviço de nuvem. Ela não é adequada para processos de longa execução. (MICROSOFT, 2014).
- Função de trabalho – Uma função de trabalho é uma função que é útil para desenvolvimento generalizado e pode executar processamento em segundo plano para uma função web. Quando você tem uma necessidade por um processo em segundo plano que execute tarefas de longa execução ou tarefas intermitentes, você deve usar esta função. (MICROSOFT, 2014)

SERVIÇOS DE DADOS

"Os serviços de dados do Windows Azure permitem às empresas armazenar, acessar, analisar e proteger seus dados e, ao mesmo tempo torná-los disponíveis a partir de qualquer lugar e em qualquer momento". (TULLOCH, 2013)

Como base para um dos requisitos necessários ao desenvolvimento do sistema CloudLU os serviços de dados do Windows Azure têm um importante papel no armazenamento e distribuição dos dados referentes ao projeto em questão, portanto dentre os vários serviços oferecidos pela plataforma, dois serviços serão amplamente utilizados, o SqlDatabase e o Storage.

SQL DATABASE

Microsoft SQL Server é amplamente utilizado como uma plataforma de dados moderna para soluções que ajudam empresas que dependem do SQL a estender seu banco de dados para a nuvem. (TULLOCH, 2013).

O serviço de dados Sql Server na nuvem foi utilizado no sistema CloudLU juntamente o *Entity Framework* que será elucidado nos capítulos seguintes, com o intuito de permitir, gerenciar e armazenar os dados resultantes dos processamentos estudados.

REST

A API Rest, com seus conceitos básicos, foi amplamente utilizada nos processos que envolvem a comunicação com os serviços de nuvem *blob storage* e *queue storage* que serão apresentados a seguir.

REST é um estilo híbrido derivado de vários padrões baseados na web e combinados com as restrições adicionais que definem uma interface de conexão uniforme. (FIELDING, 2000).

Fielding descreveu sistemas de informação como distribuídos, como a Web é construída e operada. Ele descreveu a interação entre os recursos, e o papel de identificadores únicos em tais sistemas. Ele também falou sobre o uso de um limitado conjunto de operações com a semântica uniforme para a construção de uma infra-estrutura onipresente que pode

suportar qualquer tipo de aplicação como HTTP, URL, URI e XML/HTML/JPEG/TXT. (Webber, Parastatidis e Robinson, 2010).

Rest, através das descrições dadas é um padrão de comunicação entre os serviços da web que se baseiam no protocolo HTTP. Basicamente as restrições que a descrevem, são, entre outras, relacionadas ao fato de que o conteúdo que se quer acessar em um determinado serviço na web deve ser através de um endereço único, que deverá conter todas as instruções necessárias para que uma determinada mensagem seja processada em sua URL, e/ou no cabeçalho e/ou no corpo da mensagem a qual se quer enviar e/ou receber.

Supondo um serviço que gerencie a manutenção de um cadastro de produtos pela Web, as URLs para manipular esses produtos podem ter os seguintes exemplos:

- Exemplo de uma URL REST que envia dados de um determinado produto

  POST http://www.seudominio.com/produtos

- Exemplo de uma URL REST que retorna dados de um determinado produto

  GET http://www. seudominio.com/produtos/33245

- Exemplo de uma URL REST que apaga dados de um determinado produto

  DELETE http://www. seudominio.com/produtos/33245

JSON

Assim como é o caso da Api Rest o formato JSON também foi utilizado pelas mensagens trafegam no sistema CloudLU, merecendo portanto uma breve descrição de sua estrutura.

Java Script Object Notation (JSON) é um formato de texto para a serialização de dados estruturados. É derivado de objeto literais de JavaScript, conforme definido na programação ECMAScript (ECMA, 1999). JSON pode representar quatro tipos primitivos (strings, números, valores booleanos e nulos) e dois tipos estruturados (objetos e arrays). Uma string é uma seqüência de zero ou mais caracteres Unicode [Unicode]. Um objeto é uma coleção não-ordenada de zero ou mais, com pares nome / valor. Os objetivos do projeto JSON é para que a mensagem seja mínima, portátil, textual, e um subconjunto de JavaScript. Um *parser* JSON transforma um texto JSON em uma outra representação e deve aceitar todos os textos que estão em conformidade com a gramática JSON. (CROCKFORD, 2006)

Exemplo de uma estrutura Json para uma estrutura de cadastro de produto:

```
{
    “id” :33245,
    “nome” : “Nome do Produto”,
    “preco”: 12.50
}
```

ARMAZENAMENTO BLOB

“Blob (*Binary Large Object*) fornece um mecanismo simples para armazenar grandes quantidades de dados de texto ou binários, como imagens, áudio ou arquivos visuais. Windows Azure Blob Storage pode auto alocar até 200 terabytes e pode ser acessado usando APIs de REST.” (TULLOCH, 2013).

O serviço de BLOB é baseado em três recursos: a conta blob, o recipiente, e o blob propriamente. (MICROSOFT, 2014). A conta blob, juntamente com a chave de acesso provê acessibilidade ao serviço de blob, enquanto o recipiente é comumente conhecido por diretório em uma estrutura de arquivos hierárquica, porém permite apenas um item na hierarquia, o nó raiz, e o próprio blob, podendo conter inúmeros recipientes. Já o blob é a representação binária comumente conhecida como arquivo, e pode ser acessado também através de uma única URL, com a utilização do padrão REST.

O recipiente onde é armazenado o blob pode ter dois diferentes atributos de acessibilidade: o público, que permite o acesso através de uma url sem autenticação; e o privado que já exige autenticação da qual não é objetivo tratar aqui. O blob segue os mesmos atributos de seu recipiente, porém ele pode ser armazenado diretamente na raiz, sem um recipiente associado e ter os mesmos atributos: público e privado.

Exemplo de URL utilizada para acessar um blob público:

http://SUACONTA.blob.core.windows.net/SEURECIPIENTE/SEUBLOB.txt

No sistema CloudLU, o serviço blob é o mecanismo de armazenamento dos dados, juntamente com o mecanismo de filas, que será visto a seguir, que ordena e disponibiliza tais arquivos para acesso. Veja na Figura 16 a ilustração referente à ambos mecanismos.

WINDOWS AZURE QUEUE

"As aplicações de negócios são muitas vezes de múltiplas camadas, para tanto é necessário que uma camada seja capaz de se comunicar com outra de maneira rápida segura e confiável para aplicativos implantados na nuvem. Isso pode se tornar uma questão fundamental já que a aplicação pode ser executada em servidores físicos em datacenters localizados em diferentes locais geográficos, às vezes até em continentes distintos. Windows Azure fornece várias maneiras para os diferentes componentes de uma aplicação baseada em nuvem de comunicarem eficazmente uns com os outros, entre eles o Windows Azure Queue." (TULLOCH, 2013).

O serviço do Windows Azure *Queue*, também nomeado como filas do Windows Azure, permite um ambiente de enfileiramento de mensagens que serão lidas e/ou processadas em momentos posteriores, esse mecanismo reduz o tempo de resposta para o usuário e/ou serviços, pois evita a latência de acesso a outros servidores, diminuindo então a sobrecarga destes.

No sistema CloudLU o uso de filas é um dos componentes mais importantes, pois é este mecanismo que armazena a solicitação de processamento de uma decomposição LU, e posteriormente recebe os resultados mensurados desta.

O serviço de dados *Storage* do Windows Azure (Figura 16), composto também por blob e filas, onde o *WebRole* tem o papel de encaminhar mensagens para a fila e/ou para o armazenamento blob, e o WorkerRole tem o papel de utilizar estas mensagens ou blobs para processamento.

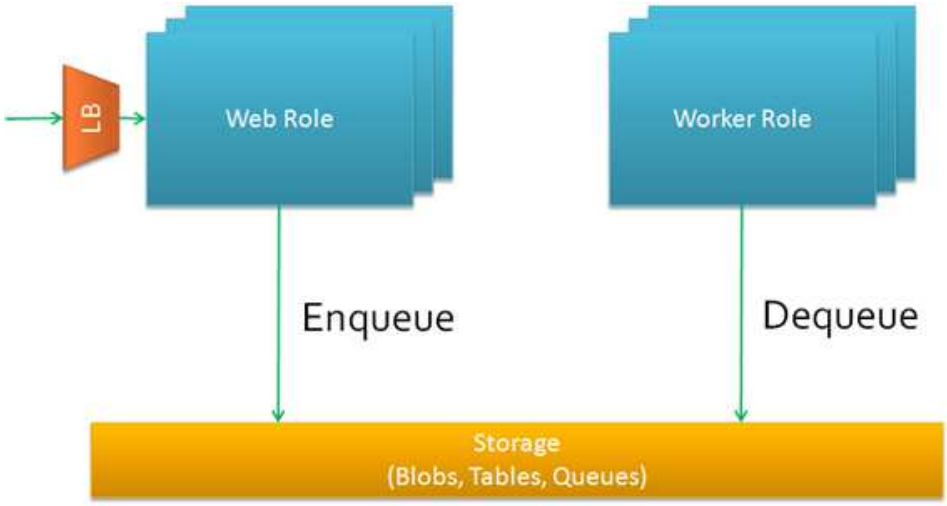

Figura 16 - Características dos serviços de dados do Windows Azure.

Fonte: adaptado de (MICROSOFT, 2014)

Projeto AzureStorageSample

O projeto AzureStorageSample é um projeto disponibilizado por (Dpallmann, 2011) que possui uma amostra de códigos que implementam as funcionalidades de acesso aos serviços do Windows Azure (blob, queue e table) com a utilização da Api Rest.

Este projeto possui exemplos de código que foram utilizados na composição da estrutura de comunicação entre os serviços de nuvem e os nós de processamento no sistema CloudLU, e foram implementados utilizando a plataforma de desenvolvimento .NET da Microsoft

APÊNDICE B

Componentes da plataforma de desenvolvimento .NET

Para a produção do sistema CloudLU foram utilizados alguns componentes da plataforma .NET, assim como alguns recursos disponíveis da linguagem Visual C#, para tanto é necessária fornecer visibilidade para estes componentes que possuem compatibilidade com tal linguagem e, que foram também parte integrante deste sistema.

Visual Studio

"O Visual Studio é um conjunto abrangente de ferramentas e serviços que ajuda você a criar uma ampla variedade de aplicativos, para a plataforma Microsoft e além." (MICROSOFT, 2014)

O Visual Studio é a ferramenta principal de desenvolvimento de todos os códigos e componentes neste projeto, a versão utilizada é a 11.0 (Ultimate 2012) e a licença deste produto foi adquirida através do programa DreamSpark por meio de um acordo existente entre a Universidade Estadual de Campinas e a Microsoft.

Visual C#

"O Visual C# é uma linguagem de programação moderna, de alto nível, de vários paradigmas e finalidades, destinada à criação de aplicativos com o uso do Visual Studio e do .NET Framework. O C# foi desenvolvido para ser simples, poderoso, fortemente tipado e orientado a objeto. As várias inovações no C# permitem um rápido desenvolvimento de aplicativos sem deixar de lado a expressividade e a elegância das linguagens no estilo do C." (MICROSOFT, 2014)

Entity Framework

"O ADO.NET Entity Framework fornece uma experiência de acesso a dados LINQ fortemente tipada nos bancos de dados relacionais incluindo acesso ao SQL Server de uma forma direta e eficiente." (MICROSOFT, 2014)

LINQ

"Uma query (consulta), é uma expressão que recupera dados de um fonte de dados. As consultas normalmente são expressas em uma linguagem especializada de consulta. Diferentes linguagens foram desenvolvidas ao longo do tempo para os diversos tipos de fontes de dados, como por exemplo SQL para bancos de dados relacionais e XQuery para XML. Portanto, os desenvolvedores tiveram que aprender uma nova linguagem de consulta para cada tipo de fonte de dados ou formato de dados que eles devem oferecer suporte. O LINQ simplifica esta situação oferecendo um modelo mais simples e consistente para trabalhar com dados em vários tipos de fontes de dados e formatos." (MICROSOFT, 2014)

Contextualizando as aplicações destes componentes, que foram necessários para facilitar e automatizar o acesso ao banco de dados SqlServer, fazendo uso da plataforma de Entidades (*Entity Framework*) juntamente com o LINQ.

Projeto ManagedCuda

"ManagedCUDA visa uma integração fácil de CUDA da NVidia em aplicações para plataformas .NET, escritos em C #, Visual Basic ou qualquer outra linguagem." (KUNZMI, 2011)

"ManagedCuda fornece um acesso intuitivo a biblioteca Cuda para qualquer linguagem da plataforma .NET" ... "Em contraste com a API de tempo de execução, managedCUDA tem uma abordagem diferente para representar detalhes CUDA: managedCuda é orientada a objeto."... "...fornecendo métodos correspondentes por classe." (KUNZMI, 2011)

Para utilizar o código gerado em CUDA, através da linguagem C++ na qual foi produzido o código de decomposição LU, o projeto ManagedCuda, forneceu a integração ideal com a plataforma .NET e consequentemente tornou viável o reaproveitamento de código com intuito de promover o desenvolvimento do sistema CloudLU de maneira rápida e fácil.

Projeto DotNetZip

"DotNetZip é uma biblioteca grátis, rápida e uma ferramenta para manipular arquivos compactados. Use VB, C# ou qualquer linguagem .NET para criar, extrair ou atualizar arquivos compactados." (CHEESO e JAANS, 2007)

No sistema CloudLU a utilização desta biblioteca de compactação de arquivos foi útil para diminuir o espaço ocupado e o tempo de transferência utilizado para armazenamento no *BlobStorage* dos arquivos que compõem a matriz de decomposição LU. Como esta biblioteca foi feita utilizando a linguagem C# ela é totalmente compatível com a plataforma escolhida, auxiliando inclusive na velocidade de transferência de arquivos do *BlobStorage* para o nós de processamento.

APÊNDICE C

Os algoritmos de decomposição LU

Neste projeto foram utilizados, três diferentes algoritmos de decomposição LU, cada um com sua abordagem, porém com o mesmo objetivo, segue detalhes da implementação do algoritmo sequencial, o algoritmo paralelo com uso de threads da cpu e em seguida o algoritmo de decomposição LU utilizando o modelo de programação CUDA, para processamento na GPU.

Algoritmo de decomposição LU sequencial

```
public void ResolveLU(float[][] A)
{
    int n = A[0].Length;
    for (int i = 0; i < n-1; i++)
    {
        ResolveMax(ref A, i);
        float _piv = A[i][i];
        for (int k = i + 1; k < n; k++)
        {
            ResolveLine(A[i], A[k], i, n, _piv);
        }
    }
}
public void ResolveLine(float[] linAi, float[] linAk, int i, int n, float piv)
{
    float _m = linAk[i] / piv;
    for (int j = i; j < n; j++)
    {
        linAk[j] = linAk[j] - _m * linAi[j];
    }
    linAk[i] = _m;
}
public void ResolveMax(ref float[][] A, int i)
{
    MaxPosition _maxPos = SolveMaxPosition(A, i);
    if (_maxPos.position != i)
    {
        float[] _toChange = A[_maxPos.position];
        A[_maxPos.position] = A[i];
        A[i] = _toChange;
        _toChange = null;
    }
}
public MaxPosition SolveMaxPosition(float[][] array, int column)
{
    int _position = 0;
    float _max = 0;
    for (int i = column; i < array.Length; i++)
    {
        if (Math.Abs(array[i][column]) > _max)
        {
            _position = i;
            _max = Math.Abs(array[i][column]);
        }
    }
    return new MaxPosition() { maxValue = _max, position = _position };
}
```

Figura 17 - Algoritmo de decomposição LU sequencial

O algoritmo de decomposição LU sequencial (Figura 17) efetua a decomposição por blocos. Após receber a matriz de elementos, o algoritmo verifica seu tamanho e inicia o primeiro laço, calcula o melhor pivote baseado na coluna “i” em que se encontra e faz a troca se necessário, no laço “k” efetua o cálculo de cada linha, baseando-se no pivote e fazendo as reduções necessárias no laço “j”.

- Algoritmo de decomposição LU com a utilização de threads

```
public void ResolveLU(float[][] A)
{
    N = A[0].Length;
    for (int i = 0; i < N - 1; i++)
    {   ResolveMax(A, i);
        float _piv = A[i][i];
        int itemsToProcess = N - 1 - i; //threads reduzidas a cada laço
        Parallel.For(i + 1, itemsToProcess, k =>
            {   ResolveLine(A[i], A[k], i, k, _piv);
            });
    }
}
public void ResolveLine(float[] linAi, float[] linAk, int i, int k, float piv)
{
    float _m = linAk[i] / piv;
    for (int j = i; j < N; j++)
    {   linAk[j] = linAk[j] - _m * linAi[j];
    }
    linAk[i] = _m;
}

public void ResolveMax(float[][] A, int i)
{
    int Alength = A.Length;
    float[] _array = new float[A.Length];
    int[] _indices = new int[A.Length];
    Parallel.For(i, Alength, k =>
    {   _array[k - i] = A[i][k];
        _indices[k - i] = k - i;
    });
    Array.Sort(_array, _indices);
    MaxPosition _maxPos = new MaxPosition()
    {   maxValue = _array[_array.Length - 1],
        position = _indices[_array.Length - 1] + i
    };
    if (_maxPos.position != i)
    {   float[] _toChange = A[_maxPos.position];
        A[_maxPos.position] = A[i];
        A[i] = _toChange;
        _toChange = null;
    }
}
```

Figura 18 - Algoritmo de decomposição LU utilizando a biblioteca Parallel

O algoritmo de decomposição LU com a utilização de threads (Figura 18) segue os mesmos padrões de laços que o código sequencial, porém, alterando-se o laço “for”, para o laço “parallel.for”, o qual utiliza threads para cada “i” item do laço, reduzindo o número de threads conforme a decomposição LU é finalizada, compondo os blocos de processamento.

Algoritmo de decomposição LU com a utilização do modelo CUDA em GPU

```
public LuCuda()
{
    ctx = new CudaContext(CudaContext.GetMaxGflopsDeviceId()); //Init Cuda context
    string resName;  //Load Kernel image from resources
    if (IntPtr.Size == 8)
        resName = "LuCuda_x64.ptx";
    else
        resName = "LuCuda.ptx";
    lud_block_scale_v2 = ctx.LoadKernelPTX(stream, "lud_block_scale_v2"); // load methods from .ptx
    lud_block_pivot = ctx.LoadKernelPTX(stream, "lud_block_pivot");
    lud_block_pivot_L2 = ctx.LoadKernelPTX(stream, "lud_block_pivot_L2");
    lud_block_swap_v2 = ctx.LoadKernelPTX(stream, "lud_block_swap_v2");
}

public void ResolveLuBlock(ref LuMatrix luMatrix)
{
    int blockDimension = 10;
    d_lu = new CudaDeviceVariable<float>(luMatrix.size);
    CudaDeviceVariable<int> d_pivot = luMatrix.pivot; // Allocate memory on GPU for LU matrix
    d_lu.CopyToDevice(luMatrix.n); // Copy matrix A vector values to the LU pointer on the GPU/device = A copy to LU (will be modified)
    int n = luMatrix.height;
    for (int i = 0; i < n; i += blockDimension)
    {   int realBlockSize = n - i < blockDimension ? n - i : blockDimension;
        decomposeLU(n, i, realBlockSize, d_pivot); //decompose the current rectangular block + include pivot of all rows k+i
    }
    d_pivot.Dispose();
    luMatrix.n = d_lu; // copy from gpu memory to main memory
}
private void decomposeLU(int M, int i, int blockDimension, CudaDeviceVariable<int> pivot)
{
    int end = i + blockDimension, threads = blockDimension;
    for (int k = i; k < end; k++) // Foreach column
    {   pivot[k] = pivotRow(M, k); // Pivoting
        swapRows(M, pivot[k], k);
        int memsize = sizeof(float) * blockDimension; // Execute LUD Block Column scaling
        threads = M-k; //set number of threads
        lud_block_scale_v2.Run(d_lu.DevicePointer, M, k, Math.Min(end, M));
    }
}
private int pivotRow(int M, int k)
{
    int threads = M-k;
    int blocks = (threads + LUDBLOCK_PIVOT_BLOCKSIZE-1) / LUDBLOCK_PIVOT_BLOCKSIZE;
    dim3 dimBlock = new dim3(LUDBLOCK_PIVOT_BLOCKSIZE, 1, 1);
    dim3 dimGrid = new dim3(blocks, 1, 1);
    CudaDeviceVariable<int> d_out = new CudaDeviceVariable<int>(dimGrid.x);
    int[] h_out = new int[dimGrid.x];
    while(blocks > 1)
    {   threads = blocks;
        blocks = threads > LUDBLOCK_PIVOT_BLOCKSIZE // Adjust the number of required blocks, for the second round
            ? (threads + LUDBLOCK_PIVOT_BLOCKSIZE-1) / LUDBLOCK_PIVOT_BLOCKSIZE
            : 1;
        dimGrid.x = (uint)blocks;
        lud_block_pivot_L2.Run(d_out.DevicePointer, d_lu.DevicePointer, M, k, M * M);
    }
    ctx.Synchronize();
    h_out = d_out;
    return h_out[0]; // The highest index is in the first location
}
private void swapRows(int M, int r1, int r2)
{
    if (r1 != r2) // Only swap if there are any difference
    {   int blocks = (M + LUDBLOCK_SWAP_BLOCKSIZE2-1) / LUDBLOCK_SWAP_BLOCKSIZE2; // setting blocks to swap
        lud_block_swap_v2.BlockDimensions = LUDBLOCK_SWAP_BLOCKSIZE2;
        lud_block_swap_v2.GridDimensions = blocks;
        lud_block_swap_v2.Run(d_lu.DevicePointer, M, r1, r2); //swap row on gpu
    }
}
```

Figura 19 - Código de chamada ao contexto de execução LU de uma GPU.

O código de decomposição LU (Figura 19) faz chamadas ao contexto da GPU através da biblioteca managedCuda. Esta biblioteca permite invocar os métodos previamente compilados em um arquivo de contexto (*.ptx), criados a partir do código de decomposição LU (OVESEN, 2014) na linguagem C++.

Este código (Figura 19) foi adaptado da lógica de (OVESEN, 2014) para que pudesse ser acessado através da plataforma .NET, com o auxilio da biblioteca disponibilizada pelo projeto managedCUDA.

O código (OVESEN, 2014) contém uma estrutura semelhante aos códigos de execução sequencial, com a diferença principal que ele é otimizado para trabalhar com threads e blocos de memória ao serem processados na GPU.

Inicialmente os contextos de processamento em Cuda são carregados em memória através do método LoadKernelPTX da biblioteca managedCuda. Em seguida os dados da matriz são copiados da memória principal para a memória da GPU, através do método CopyToDevice. A cada execução do algoritmo é definido o número de blocos e tamanho de memória que será utilizado.

Todo o processamento é feito na GPU, porém as chamadas dos métodos de processamento são todas feitas pela CPU, através do algoritmo elencado ao executar o método Run, ou seja, executa-se a decomposição de um dos blocos, a chamada é retornada para a CPU, onde solicita o processamento do pivotamento e troca das linhas da matriz pela GPU. Em seguida processa-se o bloco na GPU e assim sucessivamente, até o fim do processamento de todos os blocos da matriz. Ao final do processo a matriz de elementos resultante é copiada de volta para a memória principal.

APÊNDICE D

Estrutura da solução na ferramenta Visual Studio do sistema CloudLU

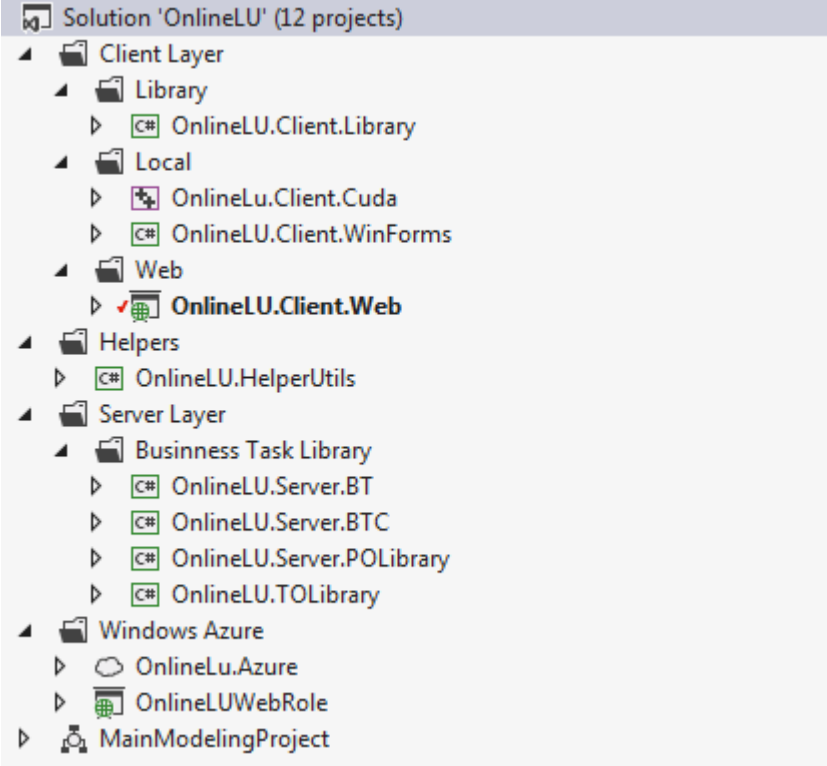

Figura 20 - Estrutura da solução do sistema CloudLU no Visual Studio 2012

A solução do sistema CloudLU foi toda definida na ferramenta Visual Studio (Figura 20), nela está a hierarquia principal das bibliotecas que compõem todo o sistema. Inicialmente foi definido um nome para a solução sempre iniciado por OnlineLU e o que se sucede após este nome na estrutura define a funcionalidade e/ou camada do projeto.

O projeto OnlineLU.Client.Library possui toda a implementação central relacionada aos acessos do sistema de gerenciamento e do nó de processamento, ou seja, toda a lógica relacionada ao processamento da decomposição LU, a fila de itens a serem compactados e submetidos e/ou recebidos do sistema na nuvem, as configurações de acesso aos serviços de nuvem, os modelos e inclusive o contexto compilado de processamento do modelo CUDA.

O projeto OnlineLU.Client.Cuda contém os códigos de processamento na GPU através do modelo Cuda, este projeto é compilado apenas uma vez e os objetos de saída de contexto são adicionados a biblioteca mencionada anteriormente.

No projeto OnlineLu.Client.WinForms, está a implementação de toda a interface do nó de processamento e suas respectivas chamadas ao projeto OnlineLU.Cliente.Library

E o projeto OnlineLU.Client.Web possui a implementação do sistema de gerenciamento na nuvem, que é referenciado internamente pelo projeto OnlineLUWebRole para que esteja coerente com as configurações exigidas para funcionamento em nuvem.

O projeto OnlineLU.HelperUtils é referenciado por todos os outros projetos, pois possui métodos utilitários necessários à comunicação entre os projetos.

O projeto OnlineLU.TOLibrary também é referenciado por todos os outros projetos por possuir a implementações de todos os objetos de transporte entre as diferentes camadas do sistema, este projeto é referenciado internamente pelo projeto MainModelingProject, pois este último é responsável pelos modelos UML, os quais geram os objetos de transporte descritos.

Os projetos OnlineLU.Server.BT, OnlineLU.Server.BTC e OnlineLU.Server.POLibrary possuem toda a implementação relacionada à persistência de dados da aplicação, o .BTC faz o controle de erros e escopos das chamadas ao EntityFramework chamando em seguida o .BT que possui a implementação da lógica de manipulação dos objetos de persistências disponibilizados pelo .POLibrary.

APÊNDICE E

Descrição geral do sistema

O sistema CloudLU tem por objetivo fornecer uma interface de gerenciamento de arquivos de decomposição LU, afim de ser disponibilizado através de sistemas em nuvem, para serem processados em nós separados geograficamente.

Em termos gerais o sistema CloudLU é subdividido em dois sistemas, sendo eles: o sistema de gerenciamento em nuvem e o sistema do nó de processamento.

O sistema de gerenciamento na nuvem é baseado na linguagem C# .NET e é construído através do modelo MVC (Controlador, modelo e visão) para funcionamento em sistema web integrados a nuvem do Microsoft Azure e é subdividido em dois subsistemas sendo eles: a função web que tem por objetivo fornecer a interface web aos usuários do sistema e a função de trabalho, que tem por objetivo executar tarefas em segundo plano.

O sistema do nó de processamento foi desenvolvido com base na linguagem C#.NET, com a utilização de Forms (formulários em inglês), adequado para uma aplicação que será instalada e utilizada em computadores desktop, laptops ou servidores.

Sistema de gerenciamento web

A Interface web do sistema de gerenciamento foi desenvolvida em linguagem Asp.NET C# e conta com um título principal: “Uma nova abordagem da programação paralela para GPU utilizando plataformas de desenvolvimento ” na parte superior, visível em todas as páginas, um menu de acesso às páginas, o corpo da página onde contém as informações inerentes a cada uma delas e o rodapé que contém a informação de direitos e informação do autor “© 2014 - Paulo Figueiredo - Faculdade de Tecnologia – UNICAMP”.

As páginas constantes no menu são: O projeto, Nova Execução, Histórico e Autor, detalhadas a seguir.

A página “O projeto” (Figura 21) faz uma breve descrição do projeto e sua aplicação, contendo dados básicos de introdução e o objetivo que o projeto busca atender, além de informações gerais sobre o propósito do trabalho.

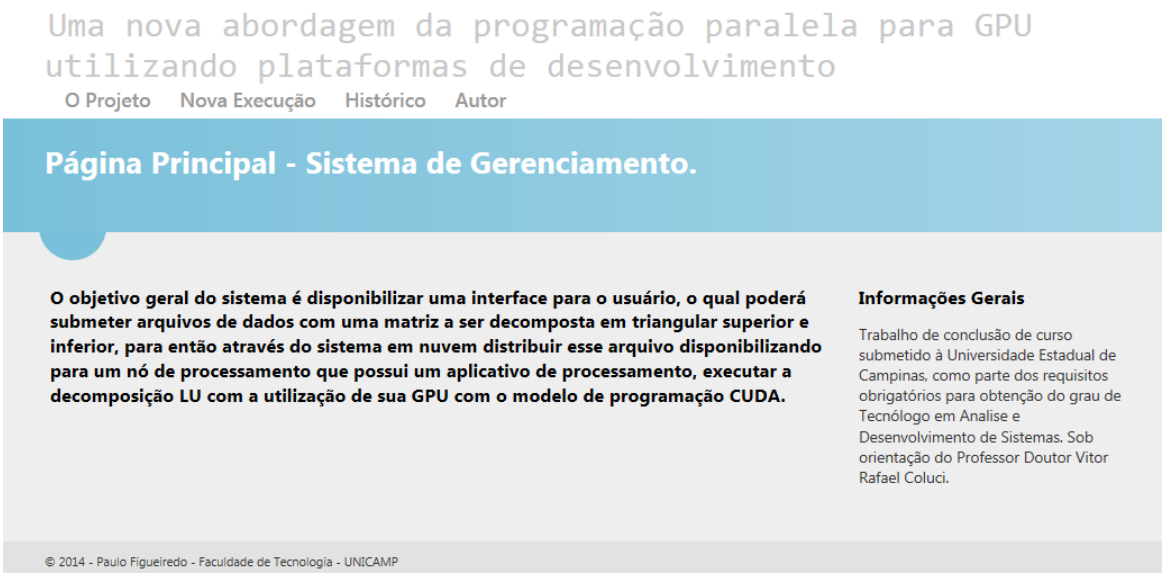

Figura 21 - Descrição da página do projeto no sistema de gerenciamento

A página “Nova Execução” (Figura 22) é composta por duas seções: Gerar nova execução e Status das execuções.

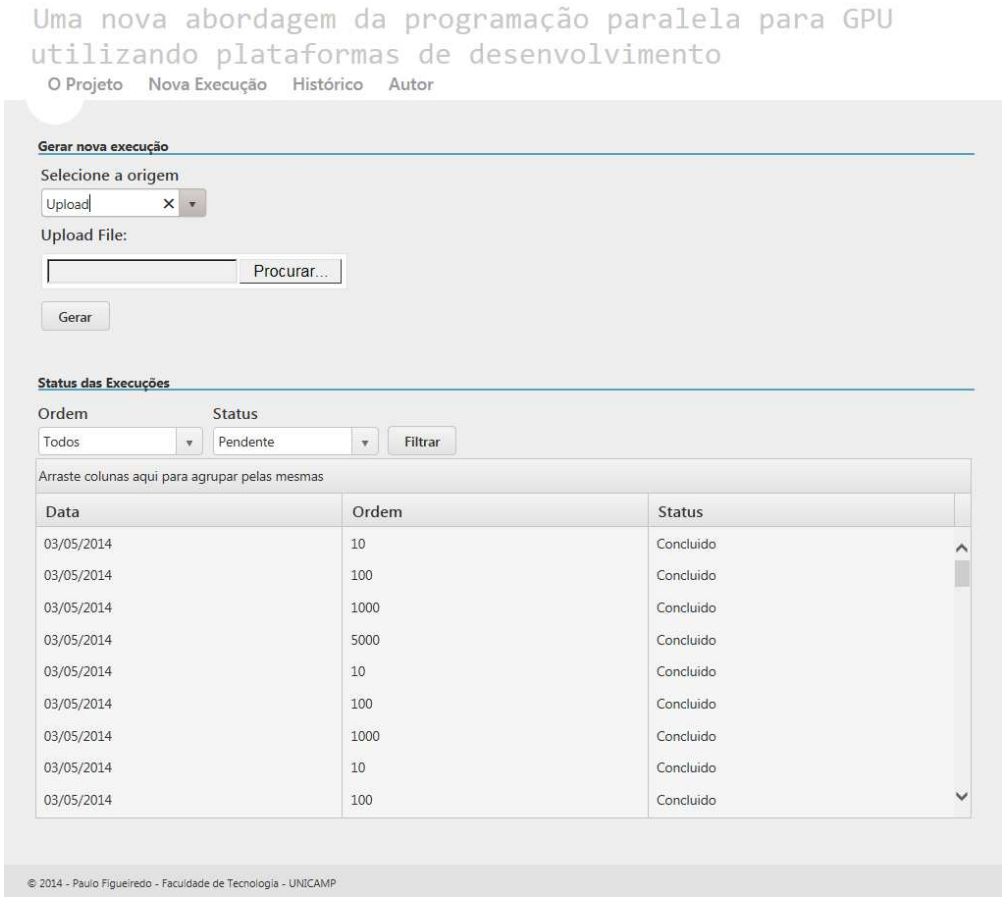

Figura 22 - A página de geração e status de projetos em execução

A primeira seção fornece um campo para buscar novos arquivos para que sejam submetidos ao processamento, com um botão que gera a submissão destes. Ao submeter um arquivo para o sistema, é exibido um status do processamento logo abaixo do campo que contém o endereço local do arquivo onde é possível mostrar mensagens de erro e/ou conclusão do envio.

A segunda seção se refere ao status destas submissões, constituída por um filtro de busca em dois campos: a ordem da matriz submetida e o status da execução, se pendente ou concluída, e uma tabela que possui as informações nas colunas da data de submissão, da ordem da matriz e seu respectivo status filtrado.

A página de histórico (Figura 23) contém o histórico de todas as execuções feitas através deste sistema relacionadas em uma tabela composta pelas colunas:

Data: exibe a data da execução;

Ordem: exibe a ordem da matriz de elementos de uma decomposição LU;

Upload(ms): exibe o tempo de transferência inicial do arquivo em milissegundos;

Download(ms): exibe o tempo do recebimento do arquivo através do nó de processamento;

Cuda (ms): exibe o tempo gasto no processamento da decomposição LU na GPU através do modelo CUDA;

Upload Resultados(ms): exibe o tempo de transferência do arquivo de resultados já processado em milissegundos;

Total(ms): exibe o tempo total gasto no processo, incluindo a soma dos tempos anteriores e outros tempos necessário ao processamento total;

Sucesso: exibe “sim” ou “não” para o status de sucesso da decomposição LU.

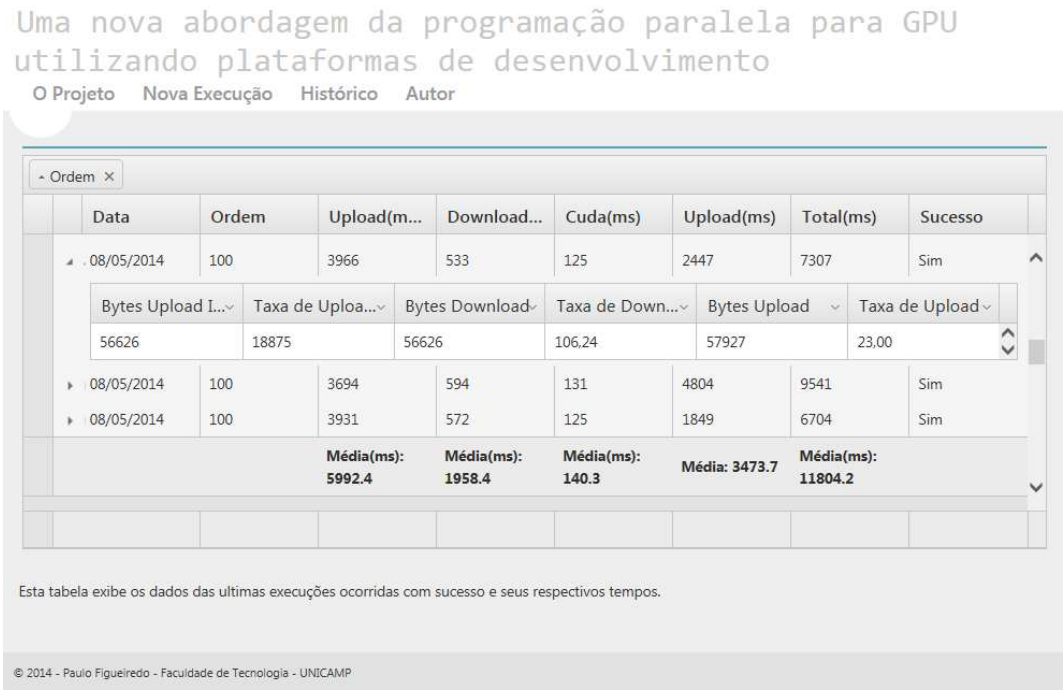

Figura 23 - Pagina de histórico contendo as execuções realizadas no sistema

Cada linha nesta tabela representa uma submissão e processamento de uma decomposição LU, assim sendo, outros dados são adicionados em uma tabela secundária, que pode ser acessada através de uma seta no inicio de cada linha. Esta tabela é composta por 6 colunas sendo:

- Bytes Upload: é a quantidade de bytes transferidos inicialmente pelo sistema;

- Taxa de Upload: é a taxa de transferência média para submeter o arquivo inicial;

- Bytes Download: é quantidade de bytes transferidos do sistema de gerenciamento para o nó de processamento;

- Taxa de Download: taxa média de download dos arquivos transferidos do sistema de gerenciamento para o nó de processamento;

- Bytes Upload Resultado: após o processamento da matriz, esta coluna representa a quantidade de bytes submetida do nó de processamento ao sistema de gerenciamento;

- Taxa de Upload Resultado: taxa média de upload dos arquivos do nó de processamento para o sistema de gerenciamento.

As linhas da tabela principal são agrupadas automaticamente pela ordem da matriz de elementos das execuções submetidas, e para cada agrupamento, exibe ao final a média das colunas de tempo mencionadas.

A página do autor (Figura 24) contém informações básicas sobre o autor deste sistema, com um mini currículo e informações de contato.

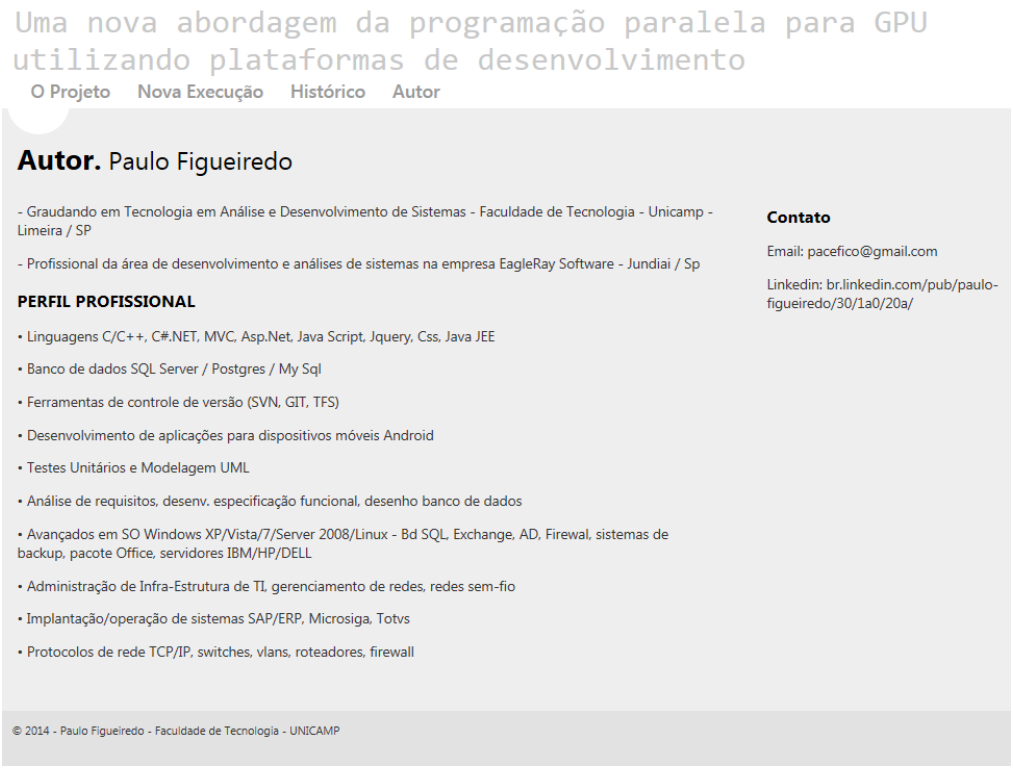

Figura 24 - Página contendo informações do autor

Sistema do nó de processamento

O sistema do nó de processamento foi desenvolvido com base na linguagem C#.NET, com a utilização de Forms (formulários em inglês), adequado para uma aplicação que será instalada e utilizada em computadores desktop, laptops ou servidores que contenham uma GPU para processamento do arquivo da matriz de elementos.

A interface deste sistema é constituída basicamente de 3 abas: Processamento, Hardware e Gerar, como segue.

A aba de processamento (Figura 25) é composta por uma área principal, onde são exibidas as informações relacionadas ao processamento em si do sistema, logo abaixo, um botão “Ativar”, que dispara o inicio da verificação de execuções disponíveis, ao lado uma caixa de seleção que permite manter o sistema sempre verificando a cada tempo, definido em milissegundos na caixa numérica ao lado. Em seguida, ao lado, possui o botão “Limpar”, que

apaga os dados da tela principal, e a caixa de seleção “Detalhes”, que permite exibir detalhes da execução do sistema.

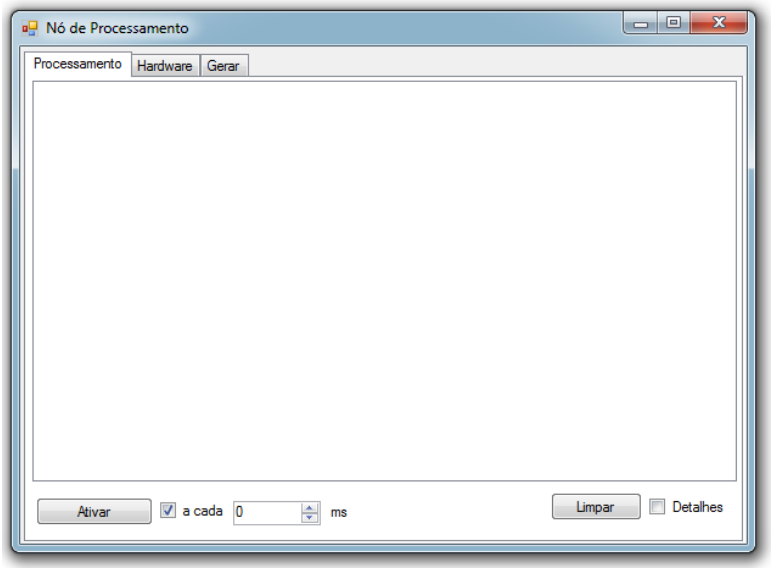

Figura 25 - Interface de processamento do sistema CloudLU

Ainda na aba de processamento, ao ativar a execução do sistema através do botão “Ativar” e tendo tarefas de execuções criadas, ele inicia o processamento, e então a área principal do sistema é preenchida (Figura 26) com os tempos e informações resumidas referentes ao processamento, que ao finalizar, terão seus resultados enviados ao sistema de gerenciamento.

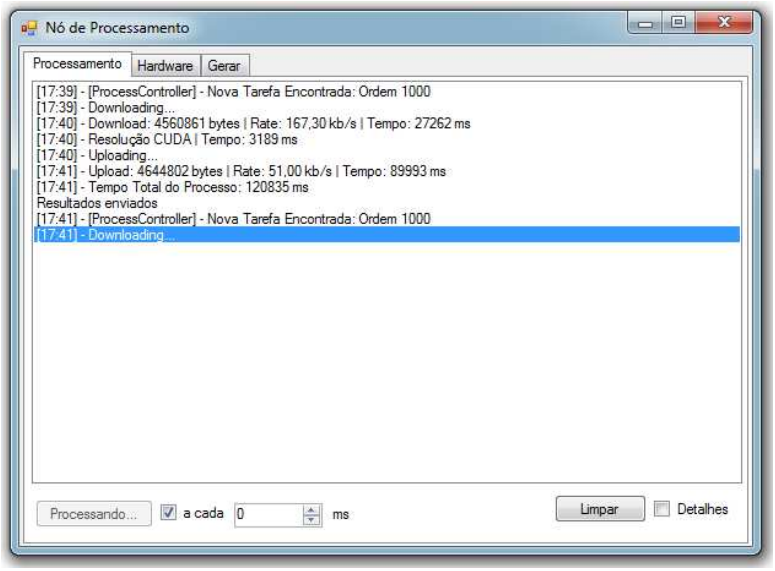

Figura 26 - Nó de processamento contendo um resumo da execução

Se a caixa de seleção “Detalhes ” estiver selecionada antes da ativação da execução, o sistema passa a exibir detalhes (Figura 27) internos do processamento, como o tempo de transferência de cada parte do arquivo de decomposição LU, a taxa de transferência desta parte, o passo que se está executando no momento e, ao final, exibe o mesmo resumo (Figura 26) com os tempos finais de execução de cada etapa.

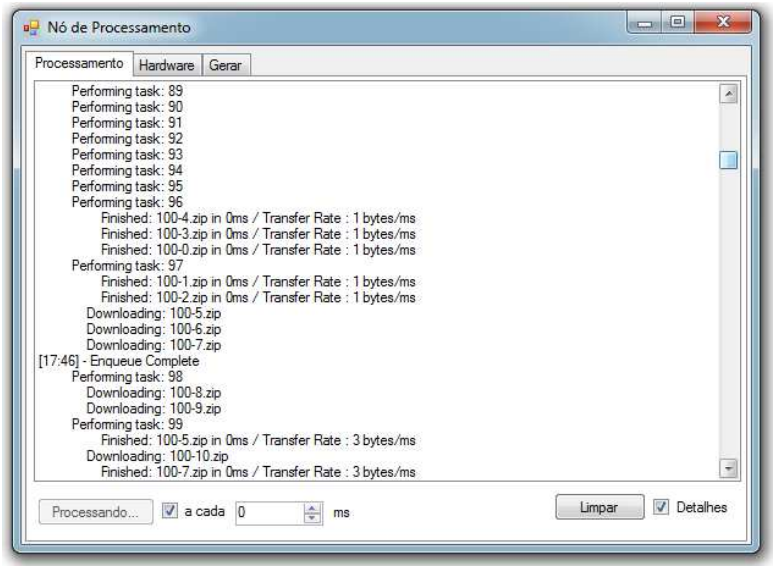

Figura 27 - Detalhes da execução do processamento.

A aba Hardware (Figura 28) detalha o equipamento em que o sistema está sendo executado, contendo: o nome do sistema, a marca e modelo da CPU, juntamente com a quantidade de núcleos de processamento, a quantidade de memória, as placas de vídeo no caso em que houver mais de uma e uma chave única de identificação do equipamento.

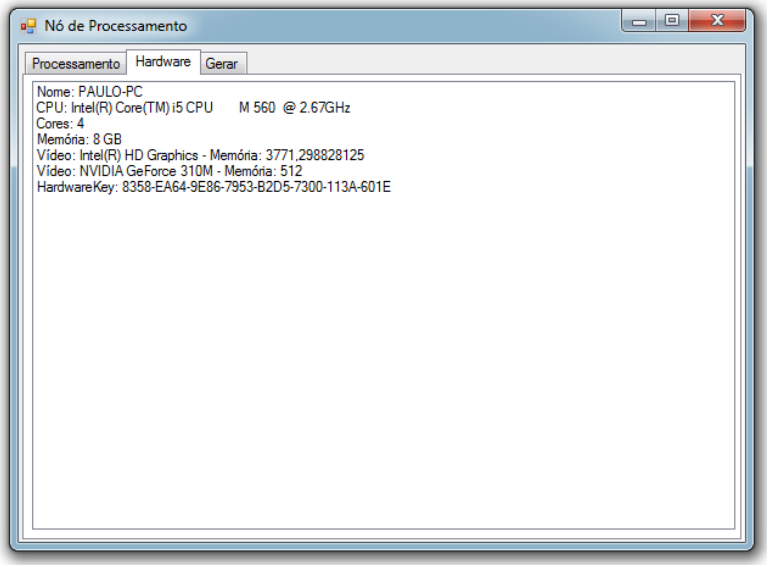

Figura 28 - Interface que indica os detalhes do equipamento onde se encontra instalado

A aba Gerar (Figura 29) permite ao usuário gerar arquivos de testes (APÊNDICE F) com números randômicos, para que possam ser submetidos através do sistema de gerenciamento. É constituída por uma caixa numérica onde se pode colocar a ordem da matriz de elementos, outra caixa numérica para selecionar o numero de casas de decimais na geração de números, uma caixa indicando onde será salvo o arquivo e, por fim, o botão Gerar, que permite disparar a criação do arquivo.

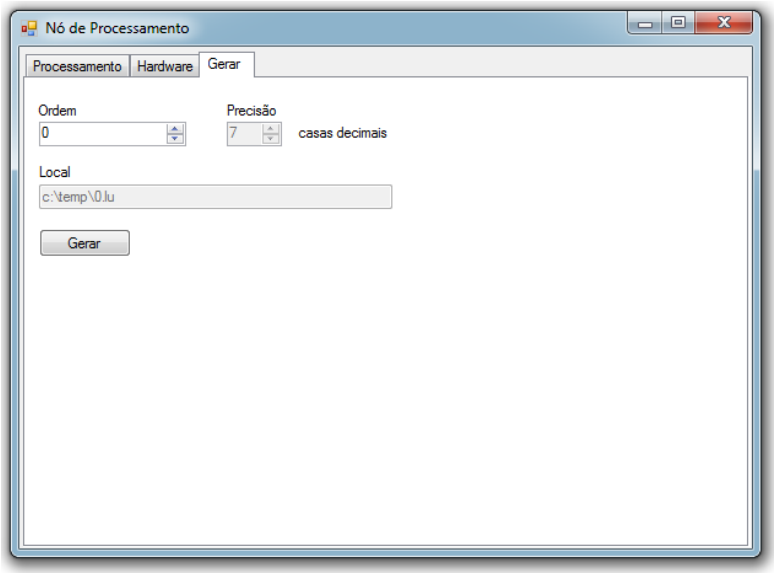

Figura 29 - Aba de geração de arquivos de testes

APÊNDICE F

Detalhes do arquivo contendo a matriz de elementos

O arquivo base utilizado para mensurar os testes de decomposição LU (Figura 30), de acordo com as limitações impostas no uso de memória para o hardware aqui utilizado e de acordo com a ordem das matrizes de teste, deve seguir o seguinte padrão:

- Conter números de -1 a 1;
- Ter no máximo 7 casas decimais, após a virgula;
- Possuir o separador “;” entre os números;
- Não possuir espaços, separadores de linha e/ou parágrafos.

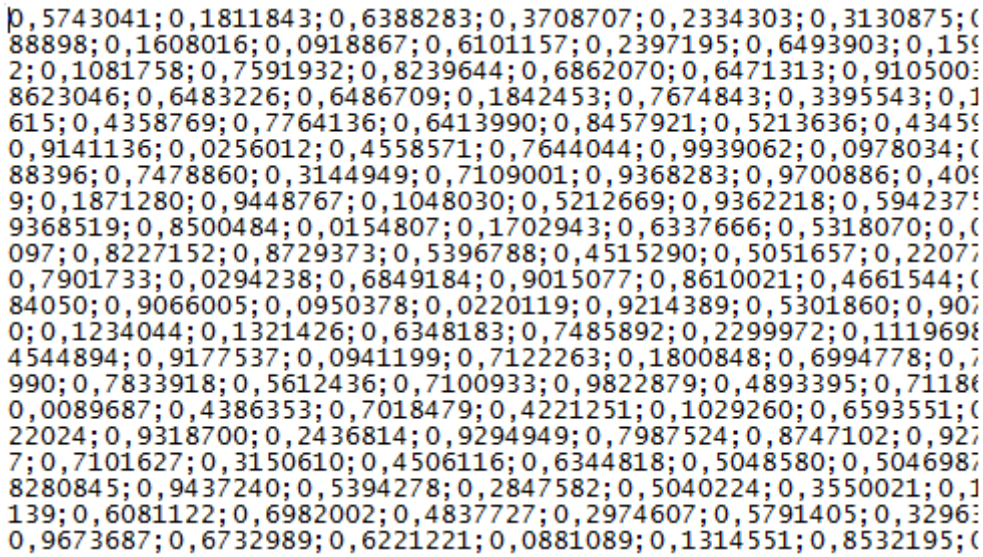

Figura 30 - Detalhes do arquivo da matriz de elementos

O tamanho em bytes do arquivo de testes é composto pela ordem da matriz, a quantidade de dígitos de cada número, mais a quantidade de casas decimais e o separadores (virgula e ponto e vírgula). Por exemplo, o número 0,5743041; (Figura 30) possui 10 caracteres no total e, em uma matriz de ordem 10, o tamanho total em bytes será de 10 x 10 (ordem) x 10 (numero com 10 dígitos), compreendendo o total de 1000 bytes.

Para gerar os números de testes para o arquivo foi utilizado a biblioteca Random da linguagem C#, e os arquivos foram gravados com a extensão .lu, aceita pelo sistema de gerenciamento.

APÊNDICE G

O processo de envio e recebimento de arquivos no sistema CloudLU

Para enviar e receber arquivos através do sistema CloudLU, foi implementado o modelo produtor/consumidor através de uma fila. Este modelo foi implementado visando a melhora na taxa de transferência de arquivos e junto, foi implementada a compactação dos arquivos.

A implementação de um modelo produtor/consumidor surgiu da necessidade de gerenciar o envio de arquivos simultâneos e com controle de erros para o *storage*.

Para facilitar o envio destes arquivos, no sistema CloudLU, o arquivo contendo a matriz de elementos foi dividido de acordo com a ordem deste, ou seja, um arquivo de ordem n = 10.000 foi dividido em 10.000 arquivos contendo 10.000 elementos.

Ao executar um laço na matriz de elementos, a cada linha desta, foi aplicado um processo de compactação com a utilização do projeto DotnetZip (Figura 31), o qual atingiu um nível de compactação de 47% para a matriz descrita.

```
public static byte[] ZipByteToByte(ref byte[] BytesToZip, int Id)
{
    byte[] BytesOutput = null;
    using (MemoryStream zipStream = new MemoryStream())
    {
        using (ZipOutputStream compressStream = new ZipOutputStream(zipStream, true))
        {
            compressStream.PutNextEntry(Id.ToString());
            compressStream.Write(BytesToZip, 0, BytesToZip.Length);
        }
        zipStream.Position = 0;
        BytesOutput = new byte[zipStream.Length];
        zipStream.Read(BytesOutput, 0, BytesOutput.Length);
    }
    return BytesOutput;
}
```

Figura 31 - Código de exemplo do projeto Dotnetzip

Após a compactação, este elemento é produzido para entrar em processamento na fila de envio, através do modelo produtor/consumidor (Figura 32). Para cada elemento uma nova thread é criada para tratar de seu envio. Durante o processamento desta thread é criado um cabeçalho baseado no modelo Rest, adaptado do projeto **AzureStorageSample**, o qual encapsula no corpo da mensagem os bytes compactados e os envia ao serviço de *storage* do Windows Azure.

```
class ProducerConsumerQueue : IDisposable
{
  EventWaitHandle _wh = new AutoResetEvent (false);
  Thread _worker;
  readonly object _locker = new object();
  Queue<string> _tasks = new Queue<string>();
  public ProducerConsumerQueue()
  {
    _worker = new Thread (Work);
    _worker.Start();
  }
  public void EnqueueTask (string task)
  {
    lock (_locker) _tasks.Enqueue (task);
    _wh.Set();
  }
  public void Dispose()
  {
    EnqueueTask (null);     // Signal the consumer to exit.
    _worker.Join();         // Wait for the consumer's thread to finish.
    _wh.Close();            // Release any OS resources.
  }
  void Work()
  {
    while (true)
    {
      string task = null;
      lock (_locker)
        if (_tasks.Count > 0)
        {
          task = _tasks.Dequeue();
          if (task == null) return;
        }
      if (task != null)
      {
        Console.WriteLine ("Performing task: " + task);
        Thread.Sleep (1000);  // simulate work...
      }
      else
        _wh.WaitOne();         // No more tasks - wait for a signal
    }
  }
```

Figura 32 - Exemplo de código de uma fila no modelo produtor/consumidor

Fonte: Adaptado de (ALBAHARI, 2014)

Implementar a fila de processamento de envio de mensagens permitiu a compactação e envio dos arquivos simultaneamente, pois enquanto um arquivo, parte da matriz de elementos estava sendo enviado, outro estava sendo compactado, gerando assim um contexto de paralelismo aplicado ao envio.

A largura de banda de internet disponível para o sistema CloudLU constituiu, em vários cenários, um gargalo para o envio de grande massa de arquivos, causando erros de tempo esgotado. Visto isto, o enfileiramento permitiu controlar o número de arquivos que poderiam ser submetidos simultaneamente, diminuindo ou eliminando, dependendo da ordem da matriz, estes erros e aumentando a taxa de transferência média, de acordo com o número de threads configuradas.

Ao comparar o número de threads simultâneas (Tabela 12) , ao utilizar uma conexão com a internet com velocidade de 10Mbps de download e 1 Mpbs de upload, pôde-se constatar que conforme aumenta o número de threads, aumenta a quantidade de erros por tempo esgotado, até dado momento em que a quantidade de erros se torna um fator limitante.

Este fator limitante se deve ao fato de que ao se esgotar o tempo limite dado a determinado arquivo em fase de envio, este arquivo é retornado ao final da fila de envio, ocupando novo tempo de processamento e também banda disponivel.

	10 threads			100 threads			200 threads		
Ordem	Erros	Tempo	Taxa kb/s	Erros	Tempo	Taxa kb/s	Erros	Tempo	Taxa kb/s
10	0	1928	0,94	0	2697	0,67	0	1815	0,99
100	0	8917	6,80	0	8644	7,01	0	8486	7,14
1000	0	58614	85,27	0	51842	96,41	0	55918	89,38
5000	22	1842405	65,04	43	983577	121,84	152	1011005	118,53
10000	30	4363988	107,59	311	3857554	121,71	2724	4406332	106,55

Tabela 12 - Número de threads no envio de arquivos simultaneamente

Portanto, na comparação feita (Tabela 12), o ideal para a conexão com a internet mencionada foi o envio simultâneo de até 100 arquivos, pois a taxa de envio obteve a melhor média dentre os cenários testados em todas as ordens da matriz de elementos.

APÊNDICE H

Banco de dados

O banco de dados da aplicação é formado por cinco entidades (Figura 33), sendo elas a tbProject, tbUser, tbHistory, tbHistoryDetail e tbHardware.

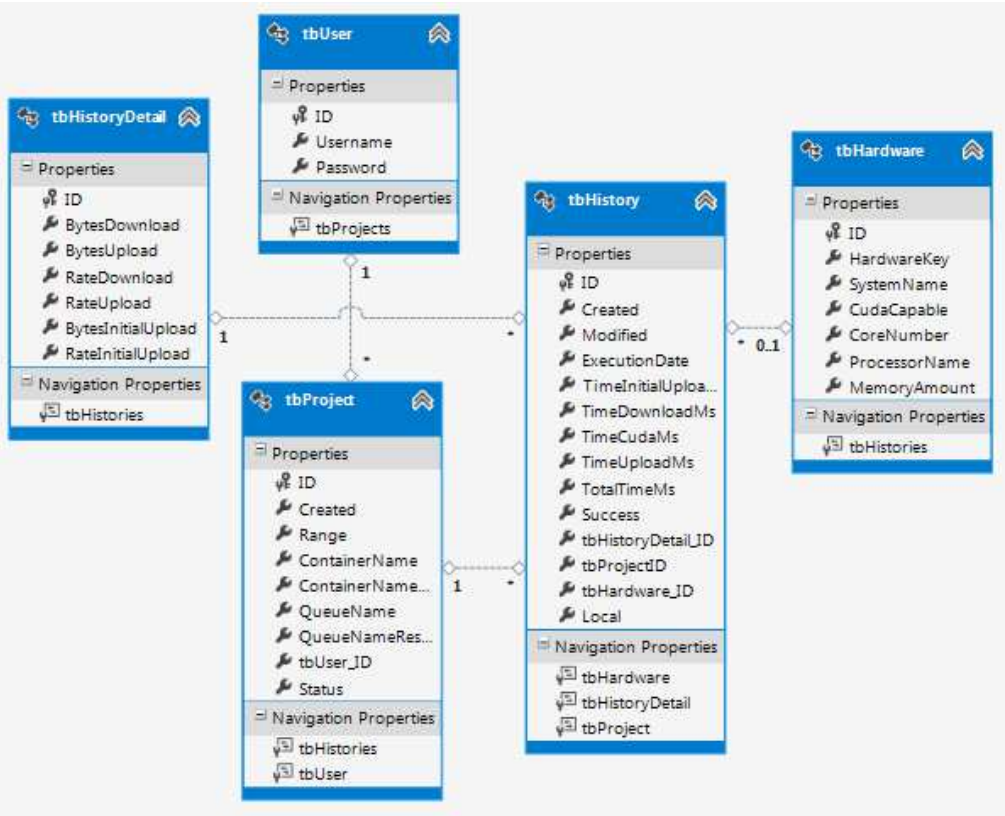

Figura 33 - Diagrama de classes das entidades de persistência do banco de dados

A tbProject é utilizada para armazenar um projeto de execução, com os dados principais da localização no storage em que os dados serão retirados e submetidos no início e fim do processamento, se relaciona com a tbUsers, para relacionar um usuário a vários projetos de execução e também permitir acesso ao sistema.

A tbHistory por sua vez se relaciona com a tbProject, contendo os dados de execução de determinado projeto uma ou mais vezes. Esta tabela armazena todos os valores mensurados na execução do projeto e se relaciona com a tbHardware, para identificar em qual equipamento determinada execução foi feita.

A tbHistoryDetailse relaciona com a tbHistory, para armazenar dados detalhados relacionados a conexão e tamanho dos dados trafegados.

Entity Framework para acesso ao banco de dados

O projeto OnlineLU.Server.POLibrary (APÊNDICE D), neste sistema é responsável por disponibilizar as entidades de acesso ao banco de dados através de objetos relacionais, que por sua vez são acessadas através do projeto OnlineLU.Server.BTC, fazendo uso destes objetos relacionais mapeados e gerenciando o contexto de acesso aos dados.

```
System.Data.EntityClient.EntityConnection _entityConnection =
        ConnectionUtils.GetTargetEntityConnection<OnlineLUEntities>();

        using (TransactionScope _scope = TransactionFactory
                        .CreateTransactionScope(transactionScope, isolationLevel))
        {

using (OnlineLUEntities _entityContext = new
        OnlineLUEntities(_entityConnection))
        {
                entityContext.ApplyIsolationLevel(IsolationLevel.ReadUncommitted);
                Type _type = typeof(HomeBT);
                MethodInfo _method = _type.GetMethod(methodName);
                HomeBT _bt = newHomeBT(_entityContext);
                _response = (MessageResponse<T>) _method.Invoke(_bt, methodArgs);
        }
}
```

Figura 34 - Exemplo de código ao utilizar entity framework

Ao realizar uma chamada ao projeto OnlineLU.Server.BTC, o sistema abre (Figura 34) uma conexão com as entidades do banco de dados, aplica diretrizes de manipulação do escopo a este contexto e disponibiliza o acesso para o projeto OnlineLU.Server.BT, representado pela classe HomeBT, realizar as devidas manipulações e retornar os dados a quem chamou.