# ParallelUS: Um Ambiente Paralelo e Distribuído para Aplicações Móveis

**Millas Avelar**[1] **, Guilherme Andrade**[2] **, Leonardo Rocha**[1]

DCOMP/UFSJ - São João del-Rei, MG , Brasil
[2] DCC/UFMG - Belo Horizonte, MG , Brasil

{millas,lcrocha}@ufsj.edu.br, gandrade@dcc.ufmg.br

***Abstract.*** *The development of new technologies has been driving significant advances in computational architectures, thus reflecting massively heterogeneous and parallel mobile architectures. Effective use of processing units on mobile devices, however, is still a challenge. In this work, we proposed ParallelUS, a parallel and distributed platform, that allows different mobile devices to cooperatively execute the same application, through task offload. We evaluate ParallelUS through different scenarios and our results show that the proposed platform provides efficient resources for implementing and executing mobile applications.*

***Resumo.*** *O desenvolvimento de novas tecnologias vem impulsionando avanços significativos nas arquiteturas computacionais, refletindo assim, em arquiteturas móveis massivamente heterogêneas e paralelas. O uso efetivo das unidades de processamento em dispositivos móveis ainda é um desafio. Neste trabalho propomos o ParallelUS, uma plataforma paralela e distribuída que, por meio do transbordo de tarefas, possibilita diferentes dispositivos móveis executar cooperativamente um mesmo aplicativo. Avaliamos o ParallelUS em diferentes cenários e nossos resultados mostram que a plataforma proposta fornece de maneira eficiente recursos para implementação e execução de aplicações móveis.*

## 1. Introdução

O desenvolvimento de novas tecnologias vem sendo caracterizada pela geração maciça de dados. Este crescimento constante aliado à necessidade de processamentos mais eficientes, vem impulsionando avanços significativos nas arquiteturas computacionais, refletindo em arquiteturas massivamente heterogêneas e paralelas. Um reflexo dessa evolução está nos dispositivos móveis. Dispositivos celulares deixaram de ser apenas aparelhos de comunicação básica entre pessoas e se tornaram provedores de diversas funcionalidades, tal como o acesso à *Internet* e *Web Services*, reprodução de vídeos em tempo real, execução de aplicações de propósito geral e várias outras aplicações que necessitam processamento computacional eficiente. Para atender essa demanda crescente de processamento móvel eficiente, é possível encontrar, em diversos dispositivos móveis populares, unidades de processamento (arquiteturas *multicore* e coprocessadores como *GPU's*) com uma alta capacidade computacional. Por exemplo, o *Samsung Galaxy S8* é equipado com um *octa-core Exynos 8895* em conjunto a uma GPU *Qualcomm Adreno 540*.

O uso efetivo dessas unidades de processamento em dispositivos móveis, no entanto, ainda é um desafio, já que são necessários esforços de programação em diferentes domínios, para explorá-las de maneira coordenada e eficiente, visando desempenho

mais alto e proporcionando uma melhor experiência ao usuário no uso de diferentes aplicações móveis. Diante disso, algumas bibliotecas e plataformas de programação, foram propostas para fornecer ferramental para auxiliar, de maneira transparente ao desenvolvedor, o uso coordenado e eficiente das unidades de processamento das arquiteturas móveis em foco [Frost 2014, Giacaman et al. 2013, Acosta and Almeida 2014, Andrade et al. 2016]. Entre essas bibliotecas e *run-time systems*, destacamos o ParallelME [Andrade et al. 2016], um mecanismo móvel e paralelo desenvolvido para explorar a heterogeneidade em dispositivos *Android*. O ParallelME destaca-se pela sua biblioteca de alto nível [de Carvalho et al. 2016], com uma abstração amigável em Java [Gosling et al. 1996], e a habilidade de coordenar de forma transparente o uso de recursos em arquiteturas móveis heterogêneas por meio de seus mecanismos de execução baseado em tarefas. Esse mecanismo de execução traduz blocos de execução, previamente definidos em Java alto-nível, para representações baixo nível, C++ em conjunto com OpenCL no nível NDK do Android. Essa representação baixo nível é obtida por meio de um compilador de código para código.

Uma limitação do ParallelME, e de outros sistemas similares, é o foco em explorar os recursos de apenas um dispositivo móvel. Essa limitação torna-se um problema quando as demandas do aplicativo móvel em execução são muito maiores aos recursos disponíveis, por exemplo, a necessidade de uma GPU dedicada, ou até mesmo capacidade de memória e bateria superiores. Nesse trabalho propomos o ParallelUS (*Parallel Universal System*), uma versão estendida do ParallelME que visa superar as limitações mencionadas acima, possibilitando que diferentes dispositivos móveis cooperem na execução de um mesmo aplicativo. A plataforma que propomos baseia-se nas características básicas do ParallelME: (i) interface de programação em alto nível e (ii) uso coordenado das unidades de processamento (CPUs e GPUs) presentes no dispositivo. No entanto, o ParallelUS vai além, possibilitando a conexão entre diferentes dispositivos para envio e recebimento de tarefas, contornando as limitações individuais do dispositivo, permitindo o uso coordenado de recursos computacionais externos e, consequentemente, aumentando o desempenho na execução de uma aplicação móvel. Nossa proposta introduz uma dinâmica computação interessante e desafiadora, desde a comunicação e conexão entre dispositivos por um esquema mestre-escravo, até mecanismos de escalonamento de tarefas entre os dispositivos (escalonamento externo) e entre unidades de processamento (escalonamento interno), tudo de forma transparente para o desenvolvedor. Podendo ser implementado em dispositivos de computação ubíqua para aumentar o poder computacional do sistema como um todo, ou utilizado para um monitoramento mais eficaz em *smart cities* através de uma rede de sensores mais autônoma e versátil.

*Enfatizamos que a concepção do ParallelUS, bem como todas as implementações e execuções de experimentos foram realizadas pelo aluno Millas Avelar, sob a orientação do professor Leonardo Rocha. O trabalho contou com a fundamental colaboração do aluno de doutorado do Programa de Pós-Graduação do DCC/UFMG Guilherme Andrade, autor e responsável pelo ParallelME, que deu todo suporte nas implementações.*

## 2. Trabalhos Relacionados

Encontramos na literatura várias propostas sobre ambientes móveis e *run-time systems* capazes de enviar tarefas para outros dispositivos, cujas discussões estão associadas aos desafios da área de *Mobile Cloud Computing (MCC)*. Uma abordagem comum

para MCC é considerar os dispositivos como provedores e consumidores de recursos, onde pares deles podem se comunicar por uma interface de rede fornecendo recursos, funcionalidades e processamento [Fernando et al. 2013]. Nesta seção destacaremos alguns sistemas [Marinelli 2009, Kemp et al. 2010, Doolan et al. 2008] que seguem a ideia do MCC, comparando-os com a concepção do ParallelUS.

**Hyrax** [Marinelli 2009] é um framework para dispositivos móveis que executa em sistemas Android que visa se beneficiar da computação distribuída, na distribuição de processamento e dados por meio da plataforma Hadoop. O Hyrax, usando Hadoop, explora o uso eficiente de conjuntos de dispositivos móveis em infraestrutura computacional para as aplicações, similar aos sistemas de computação em nuvem. Especificamente, no framework Hyrax, um servidor fixo coordena os dados e as tarefas que irão ser distribuídas para os dispositivos móveis participantes por meio de uma interface de rede isolada. Destacamos essa característica como um ponto negativo, já que se remove o conceito de mobilidade do framework e, além disso, o servidor central executa a aplicação foco, sendo os dispositivos móveis apenas nós de computação distribuída. Nossa proposta oferece uma plataforma mais versátil que o Hyrax. O objetivo do ParallelUS é permitir que qualquer dispositivo móvel execute uma aplicação e distribua processamento para outros dispositivos móveis conectados na infraestrutura ParallelUS. Além disso, por meio do ParallelUS, é possível explorar coordenadamente as diferentes unidades de processamento, CPU e GPU, presentes nos dispositivos móveis.

**Cuckoo** provê uma plataforma para os dispositivos móveis enviarem aplicações na nuvem utilizando *stub/proxy network interface*. Este sistema permite que uma aplicação possa ser enviada para processar em recursos executando uma Máquina Virtual do Java, sejam eles em clusters locais ou na Amazon Web Cloud Service (AWS). Um ponto negativo desse framework é o fato de não ser permitido que outros dispositivos móveis recebam demandas de tarefas para processar, diferente de nosso proposta ParallelUS. Além disso, as aplicações que podem ser processadas pelo Cuckook precisam suportar as execuções remotas e locais, de maneira explicita e programática. O Cuckoo oferece uma API para os desenvolvedores que abstraí o modelo tradicional de aplicação/serviço Android, separando em (i) serviços: implementação de métodos que serão enviados para a execução remota, e (ii) atividades: implementação de métodos que se relacionam à execução e interatividade local da aplicação. Diferentes versões de código são geradas pelo framework para a mesma implementação de serviços candidatos à execução externa, já que esses serviços podem ser executados em um computador com vários núcleos, por exemplo, explorando recursos paralelos.

**MMPI** apresenta uma implementação tradicional do padrão de comunicação MPI (Message Passing Interface) sobre a interface de conexão Bluetooth, para que os dispositivos possam ser consumidores e servidores de recursos computacionais. Ele fornece um ferramental para detecção e conexão de dispositivos, eliminando a necessidade do usuário/programador em se preocupar com implementações de baixo nível para conexão e gerência de conexões. Os passos do MMPI são: (i) descoberta dos aparelhos candidatos à conexão por meio de tarefas específicas; (ii) descoberta de serviços disponíveis para serem adicionados ao componente mestre; e (iii) estruturação da rede. A interface de comunicação Bluetooth apresenta limitações consideráveis, relacionados ao raio de conectividade entre dispositivos e também a perda de informações em seu envio/rece-

bimento. Esses recursos limitam consideravelmente a aplicabilidade da estrutura do MMPI. O ParallelUS oferece um ambiente completo para implementação, gerenciamento de conexão e computação de aplicativos móveis em uma arquitetura móvel heterogênea.

## 3. ParallelUS: A Parallel and Distributed System for Mobile Environment

O sistema proposto, ParallelUS (*Parallel Universal System*), é uma plataforma para computação paralela e distribuída de aplicações móveis em *smartphones* com arquiteturas heterogêneas. Ele combina uma interface de programação em alto nível com um mecanismo de execução baseado em tarefas capaz de conectar diferentes dispositivos móveis e fazer o transbordo de tarefas entre eles. Essas características superam as limitações individuais de dispositivos móveis, que pode comprometer a execução de uma aplicação onerosa em um único aparelho. A proposta do ParallelUS baseia-se no ambiente ParallelME [Andrade et al. 2016], herdando a abstração em alto nível de programação [de Carvalho et al. 2016], tal como os módulos necessários para o controle dos mecanismos de execução em baixo nível, além da dinamicidade da coordenação transparente de recursos.

De maneira geral, o ambiente proposto ParallelUS é baseado na interação entre diferentes categorias de componentes: (1) *ParallelUS Application*; (2) *ParallelUS Networkers* e (3) *ParallelUS Central Unit*. O *ParallelUS Application* é a aplicação de propósito geral implementada por meio do *ParallelUS User Library*, que explora de forma coordenada os recursos internos, bem como recursos externos de dispositivos conectados a ele, por meio do transbordo de tarefas. Para que um dispositivo seja capaz de receber tarefas e cooperar na execução de uma ParallelUS Application, é necessário executar o componente genérico ParallelUS Networker, que por sua vez, é responsável por agregar componentes Applications e Networkers componentes em uma infraestrutura controlável e coordenada. Esse componente mapeia os recursos de processamento disponíveis na rede e distribuí demandas de processamento (tarefas) provinda de ParallelUS Applications. Quando conectado, o componente *Application* pode enviar tarefas ao Central Unit, que encaminhará para um componente *Networker* conectado mais propício a executar a operação requisitada e retornar o resultado. Vale mencionar que esses componentes *ParallelUS* podem ser instalados em aparelhos Android e em computadores convencionais (tendo como pré-requisitos Java e OpenCL). Os componentes Application e Networker conectam em apenas um Central Unit que , portanto, recebe conexões de diferentes módulos de Application e Networker. A Figura 1 ilustra a interação entre os componentes ParallelUS.

### 3.1. ParallelUS Application

Com uma infraestrutura abrangente para programação paralela em ambientes móveis com suporte Java e OpenCL, o componente ParallelUS Application é composto de três principais módulos: $(i)$ ParallelUS User-library, $(ii)$ um compilador código para código e $(iii)$ um mecanismo de execução baseado em tarefas. A user-library proposta inspira-se na biblioteca de coleções Scala [Prokopec et al. 2011], que consiste em um conjunto de diferentes estruturas de dados com suporte intrínseco para computação paralela e foi inicialmente proposta no trabalho [Andrade et al. 2016]. O objetivo do ParallelUS Application é oferecer uma biblioteca orientada a estrutura de dados em Java que possa ser usada por desenvolvedores para produzir código paralelo em estruturas de baixo nível com esforço mínimo, além de fornecer uma implementação sequencial no nível do Java

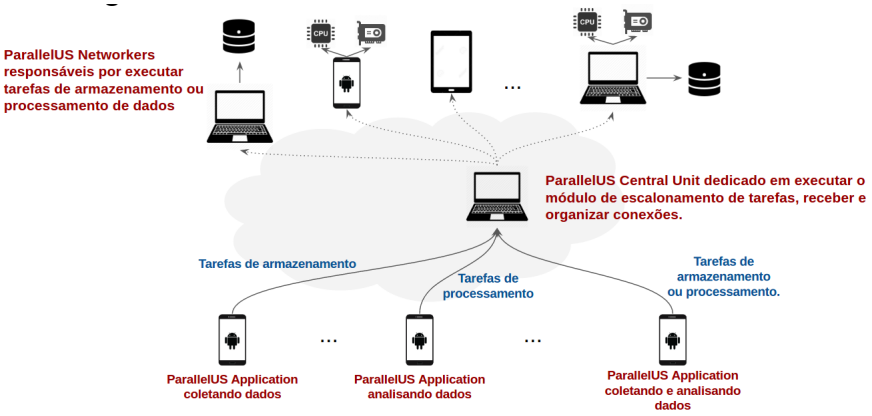


**Figura 1. Dinâmica da plataforma ParallelUS. Os componentes de aplicações enviam demandas de tarefas para uma unidade central, que coordena e escalona essa demanda entre componentes de processamento.**

SDK para fins de depuração. Nesse sentido, a user-library proposta foi projetada para suportar iterações paralelas em coleções de dados numéricos e imagens. Tais iterações são então traduzidas pelo compilador do componente ParallelUS Application para uma representação de tarefas manipuláveis pelo mecanismo de execução baseado em OpenCL e na execução distribuída e controlada em alto desempenho.

O mecanismo de execução do ParallelUS Application (run-time) foi desenvolvido sobre a plataforma baixo nível OpenCL (para permitir a execução em unidades de processamento CPU e GPU), na linguagem C++, no nível NDK do ambiente Java. É responsável por coordenar todas as unidades de processamento disponíveis na arquitetura móvel, organizar e gerenciar tarefas de baixo nível geradas pelo compilador a partir de definições expressas por desenvolvedores usando a User-Library. Esse módulo coordena a distribuição de tarefas por meio de um escalonador dinâmico de tarefas, que é capaz de encaminhar os conjuntos de processamento para uma execução interna, quanto encaminhar o envio para a central unit para que essas tarefas sejam executadas externamente.

### 3.2. ParallelUS Networker

O componente ParallelUS NetWorker é um aplicativo instalado em um dispositivo móvel que será responsável por receber tarefas, vindas da Central Unit. A ideia desta aplicação é, em primeiro lugar, conectar-se a um ParallelUS Central Unit e esperar por tarefas. Após receber e configurar a tarefa, o Networker executará usando o ParallelUS Run-time, exatamente como é a execução interna no componente ParallelUS Application. Após a execução da tarefa, os conjuntos de dados resultados do processamento são enviados à Central Unit que encaminhará para a aplicação de origem.

### 3.3. ParallelUS Central Unit

O componente ParallelUS Central Unit é responsável por agregar os demais componentes em uma infraestrutura controlável e coordenada. Esse componente mapeia os recursos de processamento disponíveis na rede e distribuí demandas de processamento e armazenamento provinda de aplicações ParallelUS. De maneira geral, todos os componentes envolvidos na execução de um contexto ParallelUS, sejam eles Networker ou Applications, se conectam na Central Unit e são constantemente monitorados, para que a variabilidade

desses componentes sejam sempre observadas e a gestão de tarefas e processamento se dê de maneira eficiente e centralizada. Todas as responsabilidades da Central Unit são separadas em módulos. O primeiro módulo, garante a gestão das conexões, armazenando de informações dos componentes conectados, além da validação e monitoramento constantes deles. Os módulos de controle de tarefas, fornece uma gestão transparente da coordenação de tarefas e buffers armazenados externamente. De forma resumida, esse módulo permite que as tarefas cheguem ao módulo de escalonamento prontas para serem distribuídas para os recursos de processamento sem, ainda, ter dependências a serem resolvidas. Por fim, o módulo de escalonamento é, basicamente, o gestor inteligente de tarefas, que distribuí o processamento entre os recursos de forma a otimizar um objetivo central do sistema. Esse módulo se beneficia de informações coletadas e monitoradas pelo módulo de conexões e, neste trabalho, constituí como o terceiro grande desafio a ser estudado e resolvido.

## 4. Avaliação da plataforma ParallelUS

A infraestrutura proposta para o ParallelUS baseia-se na comunicação entre diferentes componentes. Essa comunicação fundamenta-se no envio e recebimento de tarefas por meio de simples protocolos no modelo de comunicação por Sockets. Dessa forma, nessa seção objetivamos validar o ParallelUS na perspectiva de comunicação e funcionamento da dinâmica de envio e recebimento de tarefas. Para tanto propomos conjuntos de tarefas simuladas que mapeiam características diversas da rede em que os componentes ParallelUS estão executando, bem como características internas de tarefas, como por exemplo tamanho do buffer de entrada/saída e densidade de processamento do kernel.

### 4.1. Cenário de Avaliação

Em nossa avaliação simulamos características diversas das tarefas trocadas entre os módulos ParallelUS com o objetivo de avaliar o desempenho de comunicação e execução da infraestrutura ParallelUS implementada em diferentes perspectivas de aplicações genéricas e sintáticas. As tarefas ParallelUS impactam no desempenho da execução do sistema (comunicação e processamento) por meio de duas características: (1) Tamanho do buffer de entrada e de saída; (2) Tempo de processamento do kernel (ou do conjunto de kernel) vinculado. Para simular cenários controlados variando esses parâmetros, propomos tarefas com as seguintes característica (1.a) Tarefas Leves: Pouca quantidade de dados nos buffers de entrada e saída. Inferiores a 1kb. (1.b) Tarefas Medianas: Buffers de entrada e saída com dados entre 1mb até 10mb. (1.c) Tarefas Pesadas: Buffers de entrada e saída com dados acima de 50mb.

Considerando o esforço de processamento, propomos os cenários: (2.a) Processamento intenso dos dados: Nessas situações o tempo de execução sobre os dados de entrada de uma tarefas apresenta considerável densidade computacional. Tarefas com tempo de execução superior a 5s, por exemplo. (2.b) Processamento rápido: Nessas situações a execução do kernel é extremamente rápida em relação ao dado de entrada. Podemos destacar como exemplos, processamento de pixels de imagens para conversão em escala de cores diferentes. Esse tipo de processamento com baixa densidade computação, na maioria das vezes não faz compensar o esforço de transferência entre os componentes parallelUS.

Para simular diferentes cenários de desempenho da comunicação, considerando o impacto da rede entre os componentes ParallelUS, propomos o uso de um módulo

responsável por intermediar o envio e recebimento de mensagens com comportamentos simulados. A intuição por detrás desse módulo é simular comportamentos diversos na rede, desde atraso no envio/recebimento de mensagens de protocolo, até instabilidades e indisponibilidade de módulos ParallelUS. A Imagem 1 fornece uma ilustração do processo simulado de comunicação entre os módulos ParallelUS.

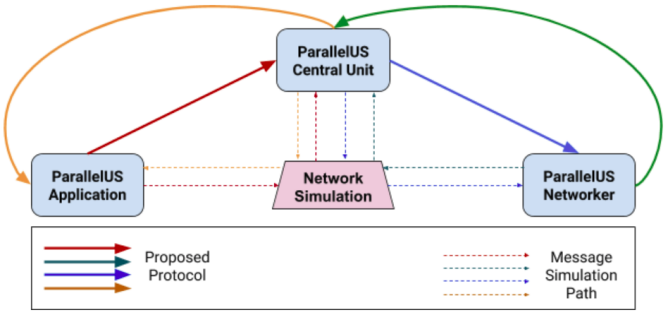


**Figura 2. A comunicação entre os componentes ParallelUS é intermediada por um módulo que simula comportamentos na transferência entre os componentes conectados, desde atrasos no processo de envio/recebimento até Instabilidade entre eles. As setas grossas e contínuas indicam qual seria o caminho direto da comunicação, e por sua vez, as setas mais finas e tracejadas indicam o caminho das mensagens nas simulações propostas.**

As características da rede que foram variadas e controladas durante os testes são:

1. Latência: tempo total de transferência de uma mensagem entre dois componentes ParallelUS. A latência é altamente dependente do volume de dados agregado na mensagem, entretanto é possível simularmos três diferentes níveis: (a) Baixa Latência: cenário em que o processo de envio e recebimento de mensagens é de 10 a 20% maior que o cenário base. Nessas situações, simulamos que a localização entre os componentes ParallelUS é próxima. (b) Latência estável: Atraso de 30% a 50% superior ao cenário baseline. Representa cenários em que os módulos ParallelUS estão em um mesmo ambiente, porém não tão próximos quanto ao cenário (a). (c) Alta Latência: Atraso no envio e recebimento superior à 60% (até 80%) do cenário baseline. Representa situações em que existe uma considerável distância entre os componentes ParallelUS conectados.
2. Disponibilidade: A característica disponibilidade nos permite simular situações de estabilidade ou instabilidade em uma rede. Uma rede altamente disponível, a taxa de perda de mensagens no processo de comunicação é bem próxima de nula, ou seja, todos os componentes do ParallelUS estão disponíveis para receber e enviar mensagens a todo instante. Por sua vez, uma rede com baixa disponibilidade a perda de pacotes é bastante alta, fazendo com que componentes ParallelUS se tornem indisponíveis durante a execução de um protocolo, sendo necessária restabelecer todos os passos de comunicação, ou simplesmente contabilizando o componente ParallelUs como inativo. A característica de Disponibilidade pode variar em dois cenários: (a) Alta disponibilidade: A taxa de perda de pacote é inferior a 5%, e em todo o processo de simulação, componentes ParallelUS não são inativados. (b) Baixa disponibilidade: A taxa de perda de pacotes é de 30 a 50%, tornando o reenvio de mensagens bastante comum, e em alguns momentos os componentes ParallelUS são inativados.

## 4.2. Avaliação da Comunicação entre Componentes

Nossa primeira avaliação valida a dinâmica de comunicação entre os componentes ParallelUS e aponta as principais características que impactam no desempenho, e para evidenciar tomaremos os casos mais extremos. Os gráficos apresentados na Figura 3 contemplam quatro cenários de rede diferentes e variações nas características das tarefas. As características das tarefas são definidas por tuplas (Peso dos dados de entrada/saída, densidade da computação dos kernels), sendo que entradas leves consideram buffers com poucos dados, e entradas pesadas consideram buffers com quantidade relevante de dados. Por sua vez, densidade de computação leve considera kernels com processamento bem rápido, e processamento denso consideram kernels com computações caras e demoradas.

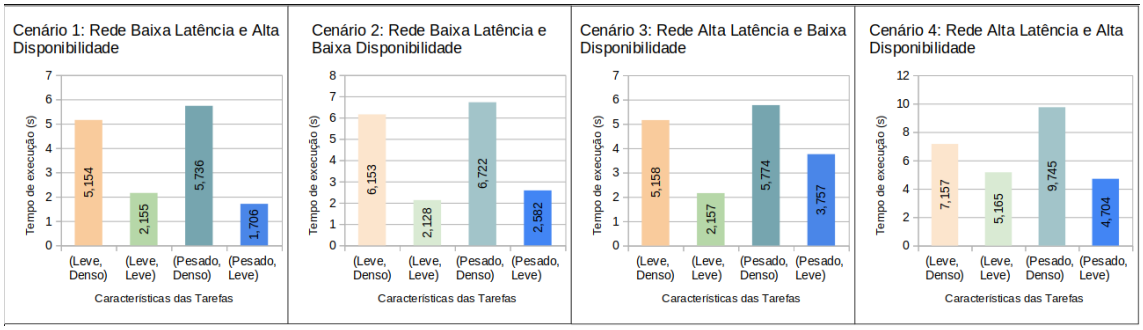


**Figura 3. Comparação entre a variação de configurações de redes e características de tarefas nos cenários simulados apresentados.**

Observando os gráficos é possível perceber que o fator ”Disponibilidade”da rede impacta diretamente no desempenho da dinâmica de comunicação entre os componentes ParallelUS. Os Cenários 1 e 4, os quais apresentam alta disponibilidade, apresentam aos menores valores para tempo de execução considerando todas as configurações de tarefas. Esse fenômeno é claramente explicável, já que, quando existe baixa perda de mensagens no processo de comunicação, as transferências são diretas e poucos reenvios são feitos. Mesmo com alta latência, o atraso no envio de mensagens é bem menor que a necessidade de reenvios constantes. Além disso, a quantidade de dados de entrada e saída impactam consideravelmente no tempo de transferência das tarefas entre os componentes ParallelUS. Dessa maneira, tarefas com grande volume de dados (entradas/saída pesadas), apresentam em todos os cenários maior tempo de execução. Esses conjuntos de tarefas são interessantes para a execução externa, quando o processamento inerente ao kernel vinculado é suficientemente intenso para compensar o esforço de transferência. Por outro lado, tarefas com volume pequeno de dados de entrada e saída (tarefas leves), são interessantes para a execução externa na maioria das situações, já que o esforço de transferência é consideravelmente pequeno. Por fim, podemos observar que a densidade de computação (tempo de execução do kernel vinculado à tarefa) possui relação direta no impacto do tempo total apresentado, sendo que para configurações (Leve,Denso) ou (Pesado, Denso) de tarefas os resultados apresentam os maiores tempo de execução.

## 4.3. Comparação entre os tipos de execução

Conforme descrito na seção 3, as tarefas ParallelUS podem ser executadas internamente no dispositivo móvel que executa o módulo ParallelUS Application, ou transbordada

para a execução externa em outro dispositivo móvel. Dessa forma, nessa avaliação objetivamos comparar a execução de tarefas internamente e externamente, apontando aqueles cenários em que a diferença entre esses tipos de execução são mais ou menos significativos, para nos dar indícios de quando a transferência de tarefas realmente contribui no desempenho total da aplicação.

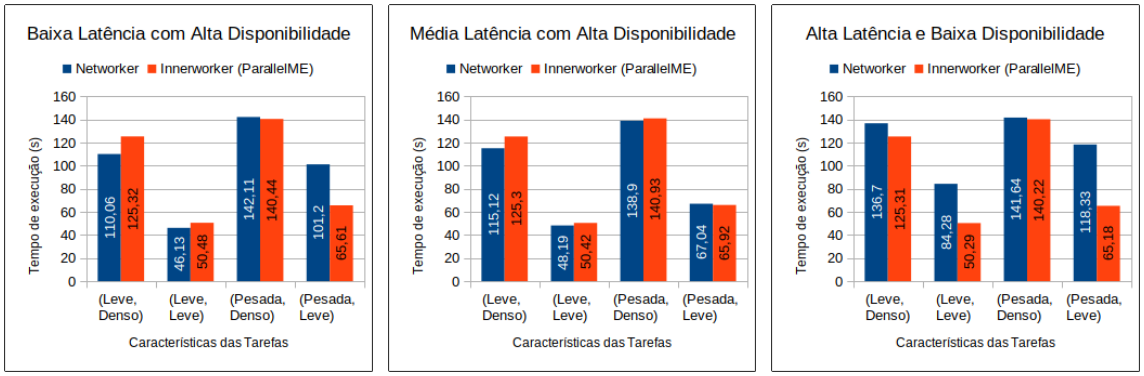


**Figura 4. Comparação entre a variação de configurações de redes e características de tarefas nos cenários simulados apresentados.**

Na Figura 4 apresentamos uma comparação entre três diferentes cenários de rede, variando as características de tarefas. Podemos observar que para os cenários de rede com Latência Baixa ou Média e Alta Disponibilidade, a transferência de tarefas é interessante para configurações em que os conjuntos de dados são leves. Característica que facilmente se explica, já que o custo de transferência de dados é mínimo, nessas situações o tempo de execução se assemelha muito à uma execução de tarefa interna (na qual explora recursos limitados de um único dispositivos). Entretanto, para conjuntos de dados pesados, a transferência de tarefas provoca um alto tempo de transferência, que impacta diretamente no desempenho da aplicação, e nessas situações a execução interna apresenta melhores resultados. Porém, quando a densidade de processamento é alta, o tempo de transferência pode ser dissolvido pelo tempo de processamento da tarefa, e em algumas situações os cenários de tarefas (Pesado, Denso) são mais interessantes para a execução externa. Por fim, no cenário de rede com Alta Latência e Baixa Disponibilidade, cenário esse com alta taxa de reenvio de mensagens e configurações, praticamente a transferência de tarefas não gera bons resultados, sendo que para qualquer configurações de tarefas, o tempo de execução da aplicação é consideravelmente maior em execuções externas.

## 5. Conclusões

Considerando os desafios relacionados às maneiras de se explorar de forma eficiente e coordenada, recursos paralelos, distribuídos e heterogêneos em sistemas móveis, propomos a plataforma ParallelUS. Nessa plataforma apresentamos componentes que provêm uma infraestrutura de implementação e geração de tarefas de processamento, os chamados ParallelUS Application, bem como os componentes que recebem e processam essa demanda, os chamados ParallelUS Networkers. Todos os componentes são geridos pela ParallelUS Central Unit, capaz de receber demandas de tarefas e distribuir de forma eficiente e coordenada entre os recursos de processamento disponíveis. Por meio da avaliação da comunicação entre Application e Networker observamos que a comunicação

na plataforma ParallelUS é válida e agrega benefícios nos casos observados, fornecendo insumos suficientes para interconectar todos os componentes propostos pela plataforma ParallelUS. Além disso, variando características de rede e tarefas, podemos observar quais são os fatores, em cada um dos cenários que impactam diretamente na performance da aplicação, apontando aqueles, em que o transbordo de tarefas não é uma prática eficiente. Como trabalhos futuros, nosso objetivo é implementar estratégias de escalonamento que coletem esses conjuntos de informações, tanto de rede quanto de tarefas, e consigam distribuir de forma eficiente os conjuntos de trabalho de uma ParallelUS Application entre a execução interna e o transbordo de tarefas.

## Referências


Acosta, A. and Almeida, F. (2014). Performance analysis of paralldroid generated programs. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE.

Andrade, G., de Carvalho, W., Utsch, R., Caldeira, P., Alburquerque, A., Ferracioli, F., Rocha, L., Frank, M., Guedes, D., and Ferreira, R. (2016). *ParallelME: A Parallel Mobile Engine to Explore Heterogeneity in Mobile Computing Architectures*, pages 447–459. Springer International Publishing, Cham.

de Carvalho, W., Andrade, G., Caldeira, P., Utsch, R., Rocha, L., Carvalho, R., and Nasser, M. (2016). Exploring heterogeneous mobile architectures with a high-level programming model. *EURO-PAR 2017*.

Doolan, D. C., Tabirca, S., and Yang, L. T. (2008). Mmpi a message passing interface for the mobile environment. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 317–321. ACM.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future generation computer systems*, 29(1):84–106.

Frost, G. (2014). Aparapi: Using GPU/APUs to accelerate java workloads.

Giacaman, N., Sinnen, O., et al. (2013). Pyjama: OpenMP-like implementation for Java, with GUI extensions. In *Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM.

Gosling, J., Joy, B., and Steele, G. L. (1996). *The Java Language Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. E. (2010). Cuckoo: A computation offloading framework for smartphones. In *MobiCASE*, pages 59–79. Springer.

Marinelli, E. E. (2009). Hyrax: cloud computing on mobile devices using mapreduce. Technical report, Carnegie-mellon univ Pittsburgh PA school of computer science.

Prokopec, A., Bagwell, P., Rompf, T., and Odersky, M. (2011). A generic parallel collection framework. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part II, EUROPAR 11*.