UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Agrupamento de Dados em GPUs NVIDIA

Uberlândia, Brasil

2023, Novembro

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Agrupamento de Dados em GPUs NVIDIA

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Daniel Duarte Abdala

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023, Novembro

Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Agrupamento de Dados em GPUs NVIDIA

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 01 de novembro de 2016:

Daniel Duarte Abdala
Orientador

Professor

Professor

Uberlândia, Brasil
2023, Novembro

Resumo

Segundo a ABNT (2003, 3.1-3.2), o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chave: Até, cinco, palavras-chave, separadas, por, vírgulas.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	*Central Processing Unit* — Unidade de Processamento Central. O principal e mais importante processador num computador. CPUs modernas possuem capacidade razoável de processamento paralelo, com dezenas de núcleos
GPU	*Graphics Processing Unit* — Unidade de Processamento de Gráficos. Um coprocessador especializado para operações vetoriais, comumente usado para operações da computação gráfica, como renderização de imagens. GPUs modernas possuem capacidade altíssima de processamento paralelo, com centenas a milhares de núcleos
VRAM	*Video Random Access Memory* — Memória de Vídeo de Acesso Randômico. Um componente das GPUs que equivale à RAM das CPUs. Uma memória volátil de alta velocidade, usada para armazenamento de dados necessários às operações gráficas realizadas pela GPU
CUDA	[inserir informação]

Sumário

1 Introdução

A busca pelo menor tempo de execução é uma diretriz ubíqua na computação. Desde os primórdios da área buscamos algoritmos e procedimentos que, dados os mesmos parâmetros de entrada, executem a mesma tarefa na menor quantidade de tempo possível. Outros recursos como espaço de memória utilizado, eficiência energética ou uso da rede em muitos cenários são mais importantes que o tempo de execução, mas ainda assim ela continua sendo um dos mais estudados parâmetros para categorização e avaliação de algoritmos e procedimentos na computação. De fato o tempo de execução — em ciclos, ou passos, de processamento — é a métrica utilizada na análise de uma das maiores incógnitas da computação, o problema P versus NP.

Um grande avanço na quantidade de poder de processamento dos computadores e, portanto, diminuição do tempo de execução de algoritmos, foi a criação dos processadores multinúcleo, permitindo a paralelização de processos. A habilidade de poder executar duas ou mais ações simultaneamente possibilitou muitos ganhos palpáveis na velocidade de execução de algoritmos e procedimentos, porém introduziu uma necessidade de mudança na forma de se pensar em resoluções de problemas computacionalmente: paralelizar um algoritmo serial (não-paralelo) não é uma tarefa trivial, e requer cuidados especiais com concorrência no acesso a recursos da máquina, interdependência de dados e cálculos, sincronização, entre outros dilemas.

Um dos componentes que mais utilizam da paralelização num computador moderno são as GPUs — unidades de processamento gráfico, ou placas de vídeo — que são basicamente processadores especializados em operações vetoriais, altamente paralelizadas, usualmente utilizadas para computação gráfica, e com sua própria memória dedicada, a VRAM. Enquanto processadores de uso geral, CPUs, costumam ter no máximo dezenas de núcleos para processamento paralelo, GPUs possuem dezenas, milhares, de núcleos para operações vetoriais.

No entanto, cada vez mais está sendo descoberto e aproveitado o potencial de uso das GPUs em atividades não apenas voltadas para renderização, interfaces e outras operações gráficas, mas sim para a computação de propósito geral. Diversos algoritmos modernos e antigos beneficiam-se imensamente do poder de alta paralelização proporcionado pelas GPUs, e com ferramentas como a biblioteca e linguagem CUDA criada pela NVIDIA, está cada vez mais fácil implementar o uso de placas de vídeo em conjunto com processadores convencionais nos mais variados algoritmos.

Nem todo algoritmo pode ser paralelizado, no entanto. Existem procedimentos e algoritmos que são inerentemente seriais (também chamados de sequenciais), como o

cálculo do n-ésimo número da sequência de *Fibonacci*, que requer que dois números prévios da sequência tenham sido calculados para obtermos o atual — salvo, é claro, alguma descoberta teórica matemática do comportamento da sequência que nos permitisse uma nova maneira de calcular o n-ésimo elemento sem essa necessidade.

É importante entender também que nenhum algoritmo é paralelizável por completo. Sempre existirão partes de algoritmos que necessariametne devem ser executadas serialmente para seu funcionamento correto. Há um limite teórico de ganho máximo que pode ser obtido ao se paralelizar um algoritmo qualquer. Esse limite é definido pela Lei de Amdahl (RODGERS, 1985): $\frac{1}{1-p}$, onde p é a razão entre tempo de execução gasto rodando código paralelizável e tempo de execução gasto no total.

E é a paralelização de uma classe de algoritmos em particular que é o foco desta pesquisa: os algoritmos de agrupamento de dados, também chamados de clusterização de dados, ou de *clustering*. Tais algoritmos, de forma sucinta, agrupam objetos de maneira que os objetos no mesmo grupo, ou *cluster*, sejam mais parecidos entre si, de acordo com alguma métrica, do que com objetos de outros grupos. A análise de clusters é essencial em diversas áreas da computação e estatística, como mineração de dados, aprendizado de máquina, compressão de dados, entre outras.

A hipótese principal deste trabalho é a de que algoritmos de clustering, em geral, são altamente paralelizáveis e apresentam um ganho considerável de desempenho (menor tempo de execução) quando implementados para utilizar o poder de paralelismo vetorial de placas de vídeo NVIDIA, através da linguagem CUDA (NVIDIA Corporation, 2018). Mais que isso, através de uma análise sistemática de estudos prévios e implementações de tais algoritmos em CUDA, visa-se generalizar o processo de paralelização destes. Isto é, identificar quais partes são necessariamente seriais, quais são paralelizáveis, e que sequência de passos gerais deve ser seguida para se conseguir paralelizar com sucesso um algoritmo de clustering qualquer e obter ganhos significativos de desempenho.

O foco de pesquisa são dois algoritmos de agrupamento especialmente notórios: o *K-means* (CUOMO et al., 2019) e o *Agrupamento Hierárquico*. Implementações e estudos realizados sobre estes foram analisados, e implementações paralelas em GPU tiveram seu desempenho comparado com as seriais em CPU.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal testar a validez de sua hipótese (discutida mais à fundo na seção 1.2) de que algoritmos de clusterização são intrinsecamente paralelizáveis, e que o ganho de velocidade ao serem paralelizados é altamente significa-

tivo.

Além disso, deseja-se compilar aqui um vasto conhecimento de como paralelizar esses algoritmos em geral, analisando principalmente os dois aqui estudados a fundo (K-Means e Agrupamento Hierárquico) e usando este aprendizado para criar um passo-a-passo genérico de como realizar tal modificação de código em um algoritmo de agrupamento qualquer.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral, é necessário completar diversos objetivos menores, ou *milestones*, antes, criando um caminho de pesquisa que foi seguido — não necessariamente na ordem apresentada. São estes:

- Pesquisar extensamente a bibliografia da área, realizando assim um levantamento do estado da arte de algoritmos paralelos de agrupamento;
- Estudar implementações já realizadas dos dois algoritmos aqui estudados, a fim de adquirir conhecimento de como a paralelização em CUDA deve ser realizada;
- Paralelizar um algoritmo de clustering "novo", isto é, nunca antes paralelizado e exibido em trabalho científico, a fim de solidificar o conhecimento e prática de programação em CUDA. Foi escolhido o algoritmo de Agrupamento Hierárquico para tal;
- Quantificar o ganho de desempenho das implementações paralelas, realizando diversos experimentos de *speedup*, usando diversos datasets de tamanhos e dimensionalidades variadas;
- Comparar o código serial (sem paralelização) com o código paralelo dos dois algoritmos analisados, extraindo assim um conhecimento de como paralelizar um algoritmo de clusterização genérico;

1.2 Hipótese

A hipótese que esta pesquisa procura testar é a de que algoritmos de clusterização em geral são inerentemente vetoriais e, consequentemente, se beneficiariam significativamente de arquiteturas de processamento vetoriais, como uma unidade de processamento gráfico, ou GPU.

Um problema ser vetorial diz respeito ao escopo de itens de dados relevantes ao problema. Grande parte dos problemas da computação são escalares, o que significa que eles lidam com itens de dados unitários, como por exemplo *integers* ou *floats*, um de cada

vez. Já um problema vetorial lida com itens de dados que são conjuntos unidimensionais, chamados vetores, que são formados por vários itens unitários de dados agrupados.

Um algoritmo que tente resolver um problema vetorial terá desempenho maior quando executado num processador vetorial, isto é, um processador que possui um conjunto de instruções capaz de manipular vetores. Apesar de um algoritmo de um problema vetorial ainda puder ser implementado e executado com sucesso num processador escalar, o desempenho será menor pois os dados vetoriais do problema terão que ter seus elementos processados um a um pelo processador, já que ele não trabalha com vetores propriamente ditos em seu conjunto de instruções.

Grande parte do ganho de desempenho supracitado vem do paralelismo proporcionado pelos processadores vetoriais, como GPUs, ao manipular conjuntos maiores de dados de uma só vez, e em vários núcleos simultaneamente, além de economizar traduções de endereço de memória e operações de obtenção (*fetch*) e decodificação (*decode*) de instruções, pois haverá um número muito menor de instruções e endereços de memória quando os dados estão agrupados em vetores, que podem ser manipulados e usados em operações como se fossem apenas um item de dados.

Este trabalho, então, visa demonstrar que algoritmos de clusterização, em geral, são intrinsecamente vetoriais. Isto é, qualquer algoritmo de clustering concebível será de natureza vetorial, pois estes analisam dados e tentam agrupá-los de a acordo com algum grau de semelhança entre eles, análise essa que pode ser feita com conjuntos dos itens de dados (vetores), ao invés de individualmente. Logo, qualquer algoritmo de clusterização teria uma parcela de seu código que seria paralelizável, e assim ganhariam desempenho significativo com uma execução numa GPU.

1.3 Justificativa

A pesquisa feita aqui pode ser de grande utilidade para a área de análise de clusters, e impulsionar a implementação de mais algoritmos paralelos de clusterização.

Com a compilação de conhecimento realizada aqui a intenção é facilitar pesquisas posteriores na área de paralelização de algoritmos de clustering e motivar novas implementações paralelas de outros algoritmos dessa classe com os experimentos de ganho de desempenho, ilustrando o quão importante é o uso de processadores vetoriais como GPUs para tornar o uso de alguns destes algoritmos prático.

Além disso, a apresentação nesse estudo de um procedimento genérico para paralelizar qualquer algoritmo de clustering será de extrema utilidade para qualquer desenvolvedor ou pesquisador que desejar implementar uma versão acelerada em GPU de um algoritmo de clusterização, mesmo este sendo totalmente novo. No mínimo servirá

de ponto de partida para o entendimento e aprendizado de como realizar tal modificação no código do algoritmo, e renderá uma implementação que serve de base para estudos e otimizações, até se atingir uma implementação digna para uso prático.

2 Fundamentação Teórica

Para compreender a pesquisa científica aqui realizada, é necessário primeiro entender o que são algoritmos de clusterização, tanto de maneira geral quanto específica, explorando os fundamentos e funcionamento dos dois algoritmos pesquisados. Veremos que a complexidade de tempo destes algoritmos tendem a ser inconvenientemente altas ($O(n \cdot \log n)$, $O(n^2)$ ou até $O(n^3)$ sendo complexidades comuns) e por isso qualquer ganho de velocidade significativo obtido valerá muito a pena.

Também é imprescindível explorar o funcionamento dos processadores vetoriais — sendo as *Unidades de Processamento Gráfico* (GPUs) o principal exemplo destes e sobre o qual essa pesquisa irá focar — e entender por quê usá-los para paralelizar algoritmos de clusterização proporcionará, em tese, um ganho de velocidade expressivo na execução destes, abrandando o peso de suas complexidades de tempo. Além de tudo isso, será apresentada brevemente a arquitetura que será utilizada para paralelizar os algoritmos estudados: a plataforma e modelo de programação CUDA, da NVIDIA, que permitirá extrair o poder de paralelização das placas de vídeo NVIDIA.

2.1 Agrupamento de dados

A clusterização, ou clustering, é a tarefa de agrupar um conjunto de elementos de modo que cada elemento de um grupo se "pareça" mais com outros elementos do grupo (cluster) que pertence do que com elementos dos grupos que não pertence — para algum significado bem definido de semelhança. É um processo muito comum e virtualmente necessário nas áreas de mineração de dados, análise estatística, análise de imagem, aprendizado de máquina, reconhecimento de padrões, e muitas outras.

O significado de um cluster não pode ser bem definido e vai depender do conjunto de dados a ser analisado e a forma que os resultados obtidos serão utilizados — de fato, esse é o principal motivo pelo qual tantos algoritmos diferentes de clustering existem (ESTIVILL-CASTRO, 2002). O comum em todas definições usadas é que um cluster é um conjunto de objetos de dados. Esses objetos são representados num espaço de dimensões iguais ao número de dados relevantes para cada objeto, e os algoritmos tentam criar grupos (clusters) nesse espaço que agrupem os objetos de uma maneira significativa, ou útil, para o estudo sendo feito e a definição de cluster sendo utilizada.

Diversos modelos de cluster podem ser usados para definir o que é um cluster: **modelos de centroide**, onde cada cluster possui um centro e os objetos pertencerão aos clusters com centros mais próximos deles, dada uma definição de distância no espaço de

dados; **modelos de densidade**, que definem clusters como regiões densas e conexas no espaço de dados, contrastando com regiões menos densas que separam os clusters; **modelos de conectividade**, que constroem clusters a partir de conexões de objetos por distância; **modelos de distribuição**, que utilizam de distribuições estatísticas, como a distribuição normal ou exponencial, para modelar agrupamentos dos objetos; entre dezenas de outros modelos. Entender o modelo de cluster utilizado é essencial para compreender um algoritmo de clusterização e as diferenças entre a multitude destes.

O resultado, ou saída, de um algoritmo de clusterização é comumente um rotulamento dos objetos de dados passados na entrada, o que indicará a divisão em clusters feita por ele. Classificações podem ser feitas quanto à natureza da clusterização obtida pelos algoritmos: ***hard clustering***, onde cada objeto pertence ou não a um cluster; ***fuzzy clustering***, onde cada objeto pertence uma certa porcentagem a cada cluster, o que pode significar, por exemplo, a chance do objeto pertencer àquele cluster (YANG, 1993). E subclassificações mais granulares ainda podem ser definidas, como: **clusterização de particionamento estrito**, onde cada objeto pertence a exatamente um cluster; **clusterização de particionamento estrito com *outliers***, onde objetos pertencem a exatamente um cluster, ou nenhum cluster, assim sendo considerados *outliers*.

2.2 Programação Vetorial

[Escrever uma sub-seção sobre Programação Vetorial]

- Tópico histórico, de onde ela vem, pra que ela serve
- Processadores que dão suporte pra operações vetoriais (hoje em dia, basicamente apenas GPUs. APUs tbm?)
- História da NVIDIA criando o CUDA
- Mudança de paradigma em relação à programação serial: escalar -> vetorial

2.3 NVIDIA CUDA

[Escrever uma sub-seção sobre programação em CUDA]

- O que é?
- Exemplos básicos de código
- Como criar um projeto cuda, onde vai cada parte (sequencial, vetorial, etc)

2.4 Algoritmo 1: K-Means

[Escrever uma sub-seção sobre o K-Means]

- Um dos algoritmos mais tradicionais, explicar a história dele
- Explicar Como ele é usado

2.5 Algoritmo 2: Hierarchical Clustering

[Escrever uma sub-seção sobre o Hierarchical Clustering]

3 Metodologia de Desenvolvimento e Pesquisa

A pesquisa realizada neste trabalho consiste de estudos e análises de trabalhos prévios, desenvolvimento de uma versão paralelizada de um algoritmo de clustering e experimentos sobre essa implementação. Pode-se dividir tal metodologia em um conjunto de etapas.

A primeira etapa consiste de uma extensa pesquisa bibliográfica. O intuito é levantar o estado da arte na área de algoritmos de clusterização acelerados em GPU usando a linguagem e livraria CUDA da NVIDIA. Entender quais algoritmos já foram implementados com sucesso em CUDA, e como foi feita tal implementação, além dos resultados (ganhos em desempenho) das mesmas. Essa etapa agregará conhecimento sobre como utilizar CUDA para acelerar algoritmos de clustering, além de dar uma ideia do tipo de ganho de desempenho esperado de uma paralelização média desse tipo de algoritmo.

A segunda etapa consiste da implementação de uma versão paralela "inédita" de algum algum algoritmo de clusterização, ou seja, paralelizar um algoritmo nunca antes paralelizado e exibido em alguma pesquisa. Usando o aprendizado adquirido na primeira etapa de pesquisa, um algoritmo será paralelizado em CUDA, e seus resultados comparados com os resultados da versão serial (rodando somente numa CPU) para garantir corretude. Os ganhos de velocidade da versão acelerada em CUDA será então comparada também com os ganhos obtidos nos trabalhos analisados na primeira etapa. Isso servirá como uma medição da efetividade da implementação feita.

A terceira etapa consiste da busca de um procedimento geral para paralelizar um algoritmo de clustering genérico. Ou seja, encontrar um passo-a-passo de modificações ao código de um algoritmo serial que, ao fim, o transforme numa versão acelerada usando CUDA desse mesmo algoritmo, ainda mantendo sua corretude e proporcionando algum ganho de desempenho.

A quarta etapa, por fim, se trata de diversos experimentos de ganho de velocidade, ou *speedup*, do algoritmo que teve aqui sua versão acelerada em GPU implementada e apresentada. Esses experimentos irão dar uma ideia do quão significativo foi o ganho de desempenho ao paralelizar o algoritmo usando CUDA, se há um teto ou chão para tais ganhos, como o *speedup* aumenta ou diminui com o aumento do dataset a ser analisado, assim como realizar uma análise de outros parâmetros importantes que não sejam velocidade, como uso de memória — afinal, as VRAMs das placas de vídeo são comumente mais limitadas em tamanho que as RAMs utilizadas pelas CPUs.

4 Experimentos e Datasets

5 Resultados

6 Conclusões e Trabalhos Futuros

7 Desenvolvimento

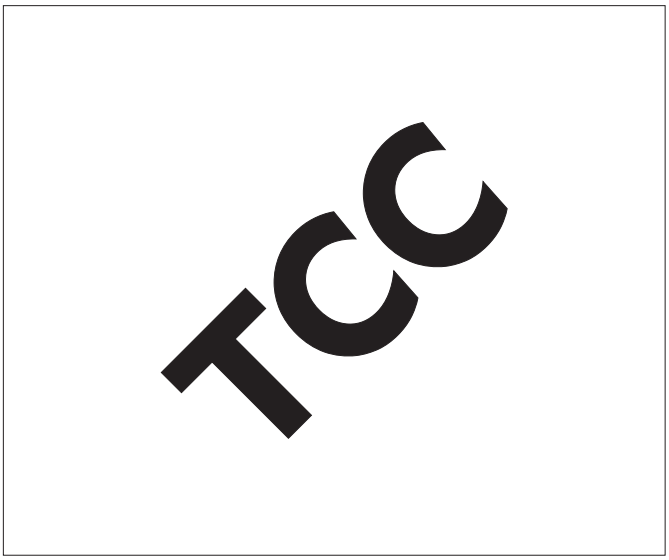

Figura 1 – Imagem de exemplo.

8 Conclusão

E daí?

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6028**: Resumo - apresentação. Rio de Janeiro, 2003. 2 p. Citado na página 3.

CUOMO, S.; De Angelis, V.; FARINA, G.; MARCELLINO, L.; TORALDO, G. A GPU-accelerated parallel K-means algorithm. **Computers and Electrical Engineering**, v. 75, p. 262–274, 2019. ISSN 00457906. Citado na página 9.

ESTIVILL-CASTRO, V. Why so many clustering algorithms. **ACM SIGKDD Explorations Newsletter**, ACM, v. 4, n. 1, p. 65–75, jun 2002. ISSN 19310145. Disponível em: <http://portal.acm.org/citation.cfm?doid=568574.568575>. Citado na página 13.

NVIDIA Corporation. **CUDA Zone**. 2018. Disponível em: <https://developer.nvidia.com/cuda-zone>. Citado na página 9.

RODGERS, D. P. Improvements in Multiprocessor System Design. **Conference Proceedings - Annual Symposium on Computer Architecture**, ACM, New York, NY, USA, v. 13, n. 3, p. 225–231, 1985. ISSN 01497111. Disponível em: <http://doi.acm.org/10.1145/327070.327215>. Citado na página 9.

YANG, M. S. A survey of fuzzy clustering. **Mathematical and Computer Modelling**, Pergamon, v. 18, n. 11, p. 1–16, dec 1993. ISSN 08957177. Disponível em: <https://www.sciencedirect.com/science/article/pii/089571779390202A>. Citado na página 14.

Parte I

Anexos

ANEXO A – Eu sempre quis aprender latim

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

ANEXO B – Coisas que eu não fiz mas que achei interessante o suficiente para colocar aqui

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

ANEXO C – Fusce facilisis lacinia dui

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.