

LIBEREZ LE POUVOIR DU BIG DATA ET DU MACHINE LEARNING

Comment les librairies Intel® Performance de Intel rendent tout ceci possible

Vadim Pirogov et Ivan Kuzmin, *Responsables génie logiciels;*
Et Sarah Knepper, *ingénieur en développement logiciels, Intel Corporation*

Nous vivons dans un monde où les êtres humains s'appuient de plus en plus sur les ordinateurs pour résoudre une variété de problèmes d'ingénierie, allant de la prévision du temps à la découverte de médicaments vitaux. Nous sommes à la veille d'un changement spectaculaire où les machines seront capables d'atteindre et même de dépasser les êtres humains dans leur capacité à prendre des décisions et à résoudre des problèmes complexes. Les ordinateurs battent déjà les meilleurs joueurs humains sur **Jeopardy*** et **Go***, et des voitures autonomes roulent déjà sur les routes de Californie. Tout ceci est possible avec les puissances de calcul petaflop (grâce à la loi de Moore) et des nombreuses quantités de données disponibles pour entraîner les algorithmes d'apprentissage automatique (machine learning).

Chez Intel, nous travaillons en étroite collaboration avec des grands établissements universitaires ainsi que des industriels dans le but résoudre les défis de l'architecture matérielle et logicielle des futures plateformes de calcul multi/many-core de Intel. Pour aider les innovateurs à s'attaquer à la complexité de l'apprentissage automatique, nous mettons à disposition des développeurs des moyens d'optimisation de performances via des outils logiciels Intel® populaires, notamment au travers de la bibliothèque Intel® Data Analytics Acceleration Library (Intel® DAAL) et des améliorations apportées à la bibliothèque Intel® Math Kernel Library (Intel® [MKL](#)).


Le défi de l'apprentissage automatique

Au cours de la dernière décennie, l'apprentissage automatique a connu une croissance extrêmement rapide. Cette croissance est alimentée par l'Internet, qui génère d'immenses quantités de données. Le désir d'extraire des modèles de ces données et d'appliquer toutes ces connaissances pour effectuer des prédictions a entraîné le développement de nouvelles approches et d'algorithmes. La croissance exponentielle de la puissance de calcul a permis de mettre en œuvre ces algorithmes sur d'énormes ensembles de données et de faire des prévisions utiles.

Les réseaux de neurones profonds (Deep Neural Networks - DNN) sont à la pointe du domaine de l'apprentissage automatique. Ces algorithmes, qui ont bénéficié d'une large adoption par l'industrie à la fin des années 1990, ont d'abord été appliqués à des tâches telles que la reconnaissance d'écriture sur les chèques bancaires. Les réseaux de neurones profonds furent un succès sans égal vis à vis de cette tâche, égalant et dépassant même les capacités de l'humain. Aujourd'hui, les DNNs sont utilisés pour la reconnaissance d'image, de vidéo, le traitement du langage naturel, ainsi que pour résoudre des problèmes complexes de compréhension visuelle tels que ceux posés par la conduite autonome. Les DNNs sont très exigeant en termes de ressources de calcul et le volume de données qu'ils doivent traiter. Pour mettre ceci en perspective, la topologie moderne de reconnaissance d'image AlexNet prend plusieurs jours pour s'entraîner sur des systèmes de calcul modernes et utilise un peu plus de 14 millions d'images. S'attaquer à cette complexité nécessite des éléments constitutifs bien optimisés pour réduire le temps d'apprentissage et de répondre aux besoins de l'application industrielle.

Intel MKL

Le [MKL](#) de Intel est une bibliothèque mathématique à haute performance pour architectures Intel et autres architectures compatibles (**Figure 1**). Cette bibliothèque fournit des implémentations de routines d'algèbre linéaires communes denses (BLAS et LAPACK) et dispersées (Sparse BLAS et Intel® MKL PARDISO), des transformées de Fourier discrètes, des mathématiques vectorielles et des fonctions statistiques optimisées pour les processeurs Intel® actuels et futurs. Intel MKL s'appuie sur un parallélisme au niveau de l'instruction, du thread et du cluster pour augmenter les performances de nombreuses applications scientifiques, techniques et financières sur les postes de travail, les serveurs et les supercalculateurs.



	Intel® Xeon® Processor 64-bit	Intel® Xeon® Processor 5100 series	Intel® Xeon® Processor 5500 series	Intel® Xeon® Processor 5600 series	Intel® Xeon® Processor E5-2600 v2 series	Intel® Xeon® Processor E5-2600 v3 series v4 series	Future Intel® Xeon® Processor ¹	Intel® Xeon Phi™ x100 Coprocessor (KNC)	Intel® Xeon Phi™ x200 Processor & Coprocessor (KNL)	Future Intel® Xeon Phi™ (KNH)
Up to Core(s)	1	2	4	6	12	18-22	TBD	61	72	TBD
Up to Threads	2	2	8	12	24	36-44	TBD	244	288	TBD
SIMD Width	128	128	128	128	256	256	512	512	512	TBD
Vector ISA	Intel® SSE3	Intel® SSE3	Intel® SSE4- 4.1	Intel® SSE 4.2	Intel® AVX	Intel® AVX2	Intel® AVX-512	IMCI 512	Intel® AVX-512	TBD

Product specification for launched and shipped products available on ark.intel.com. All dates and products specified are for planning purposes only and are subject to change without notice. 1. Not launched or in planning.

1 Processeurs ciblés par la bibliothèque Intel® Math Kernel Library (Intel® MKL)

Bien que Intel MKL ait été conçu pour le calcul haute performance (HPC), ses fonctionnalités sont aussi universelles que les mathématiques. Les fonctionnalités telles que la multiplication matricielle, la transformation de Fourier rapide (FFT), ou l'élimination gaussienne créent non seulement le fondement de nombreux problèmes scientifiques et techniques, mais aussi les bases des algorithmes de l'apprentissage automatique.

Intel MKL 2017 inclut une fonctionnalité optimisée pour profiter des algorithmes d'apprentissage automatique clés, avec de nouvelles extensions DNN afin de répondre aux besoins de calcul unique de l'apprentissage automatique. Examinons deux de ces cas : Les extensions de réseaux de neurones profonds (DNN) et les améliorations de la multiplication matricielle.

Accélérer les DNNs avec Intel MKL

Primitives DNN dans Intel MKL

L'apprentissage profond (deep learning), une branche de l'apprentissage automatique qui utilise des graphes profonds avec plusieurs couches de traitement pour modéliser des abstractions de haut niveau, est en train de conquérir rapidement les centres de données. Cette approche a été inspirée par la façon dont les organismes vivants perçoivent la réalité au travers du cortex visuel. Elle a connu un succès considérable dans les applications telles que la reconnaissance d'images et de vidéos, le traitement du langage naturel et les systèmes de recommandation. Ces charges de travail s'appuient sur de nombreux algorithmes, notamment des convolutions multidimensionnelles et des multiplications matricielles. Bien que les convolutions puissent être exprimées sous forme de multiplications matricielles, il est parfois plus efficace de mettre en œuvre une approche directe qui produit des performances nettement meilleures sur les architectures modernes.

En plus de la convolution, les charges de travail de l'apprentissage profond comprennent plusieurs types de couches qui opèrent sur des matrices à petites dimensions. Pour minimiser la surcharge des transformations de données, nous avons introduit des implémentations optimisées de ces fonctions clés dans Intel MKL 2017 dans le nouveau domaine DNN (Deep Neural Networks - Réseaux de Neurones Profonds).

Intel MKL 2017 présente le domaine DNN, qui inclut les fonctionnalités nécessaires pour accélérer les topologies de reconnaissance d'image les plus populaires, y compris AlexNet, VGG, GoogLeNet et ResNet.

Ces topologies DNN s'appuient sur un certain nombre d'éléments constitutifs standard, ou primitives, qui opèrent sur des données sous forme de jeux multidimensionnels appelés tenseurs. Les primitives comprennent la convolution, la normalisation, l'activation et les fonctionnalités internes du produit, ainsi que les fonctions nécessaires pour manipuler les tenseurs. La réalisation de calculs efficaces sur des architectures Intel doit tirer parti des instructions SIMD via une vectorisation et de plusieurs noyaux de calcul via le threading. La vectorisation est extrêmement importante, puisque les processeurs modernes opèrent sur des vecteurs de données pouvant aller jusqu'à 512 bits de long (16 chiffres) et peuvent effectuer jusqu'à deux opérations de multiplication et d'addition (fused multiply-add, ou FMA) par cycle. L'utilisation de la vectorisation requiert que les données soient situées consécutivement en mémoire. Étant donné que les dimensions typiques d'un tenseur sont relativement faibles, la modification de la disposition des données génère des surcharges importantes. Nous nous efforçons d'effectuer toutes les opérations dans une topologie sans modifier la disposition des données de primitive à primitive.

Intel MKL fournit des primitives des opérations les plus utilisées implémentées pour une disposition des données optimisé pour la vectorisation :

- Convolution directe en lot
- Produit intérieur
- Pooling (mise en commun) : Maximum, minimum, moyenne
- Normalisation : Normalisation de réponse locale (Local Response Normalization LRN) sur l'ensemble des canaux, normalisation par lot
- Activation : Unité linéaire rectifiée (Rectified Linear Unit ReLU)
- Manipulation de données : Transposition multidimensionnelle (conversion), scission, concaténation, somme et échelle

Blog - Points essentiels

Vous avez appris ceci à partir d'une machine ?

Par [LENNY TRAN](#) >

Des scientifiques, des développeurs et des chercheurs utilisent l'apprentissage automatique pour obtenir des connaissances antérieures hors de leur portée. Des programmes qui apprennent par expérience les aident à découvrir comment fonctionne le génome humain. Des programmes qui comprennent le contexte derrière le comportement des consommateurs peuvent élaborer des recommandations en temps réel. Des programmes peuvent être entraînés à reconnaître et à identifier des images et même des visages, afin d'améliorer la sécurité.

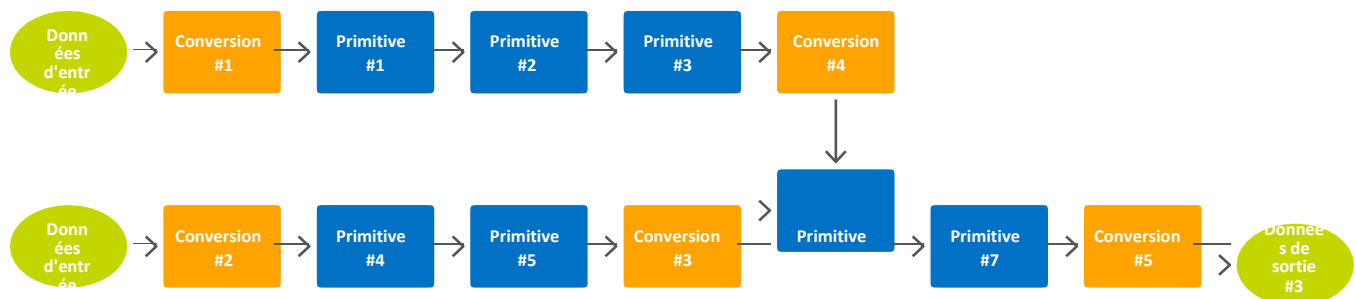
Pour en savoir plus



< L'univers parallèle

Le flux d'exécution de la topologie en réseau de neurones comprend deux phases : la configuration et l'exécution. Au cours de la phase de configuration, l'application crée des descriptions de toutes les opérations DNN nécessaires pour implémenter la notation, l'apprentissage ou d'autres calculs spécifiques à l'application. Pour transmettre les données d'une opération DNN à la suivante, certaines applications créent des conversions intermédiaires et allouent des tableaux temporaires si la disposition des données d'entrée et de sortie appropriée ne correspond pas. Dans une application typique, cette phase s'effectue une seule fois et elle est suivie de plusieurs phases d'exécution où se produisent les véritables calculs.

Au cours de l'étape d'exécution (**Figure 2**), les données sont transmises au réseau dans une disposition simple comme le NCWH (lot, canal, largeur, hauteur) et sont converties dans une disposition compatible SIMD. À mesure que les données se propagent entre les couches, la disposition des données est préservée et les conversions sont effectuées lorsqu'il est nécessaire d'effectuer des opérations qui ne sont pas prises en charge par l'implémentation existante.



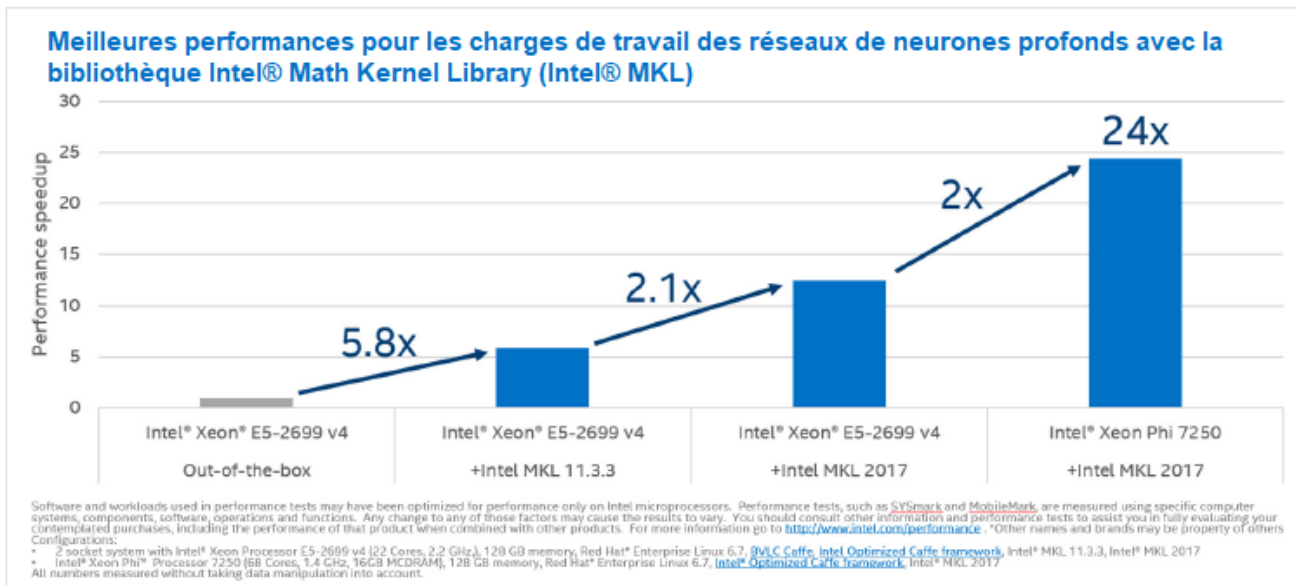
2 Étape d'entrée

Étude de cas : Caffe*

Caffe*, framework d'apprentissage profond développé par Berkeley Vision and Learning Centre (BVLC), est l'un des framework communautaires les plus populaires en termes de reconnaissance d'image. Avec **AlexNet**, une topologie de réseau neuronal de reconnaissance d'image, et **ImageNet TM**, une base de données d'images étiquetées, Caffe est souvent utilisé comme référence.

Caffe profite déjà des routines mathématiques optimisées fournies dans l'Intel MKL en utilisant l'interface standard BLAS. Cependant, l'implémentation d'origine n'utilise pas efficacement le parallélisme et ne profite pas pleinement de la fonctionnalité GEMM dans l'implémentation de la convolution. En restructurant le flux de calcul pour utiliser GEMM sur des matrices plus grandes et en extrayant un parallélisme supplémentaire avec l'aide d'OpenMP*, nous avons réalisé une multiplication de l'accélération par 5,8 pour l'apprentissage de la topologie AlexNet à l'aide d'un processeur système v4 à 2 sockets Intel® Xeon® E5-2699. Comme mentionné précédemment, une mise en œuvre de convolution directe est plus efficace. En utilisant les nouvelles primitives Intel MKL 2017, nous obtenons une multiplication de l'accélération par 11.

L'architecture Intel et Intel MKL continuent de se développer. En juin, nous avons lancé le processeur Intel® Xeon Phi™ série x200 avec une architecture massivement parallèle comportant jusqu'à 72 cœurs, des extensions vectorielles avancées Intel® 512 (Intel® AVX-512) et une mémoire rapide sur puce. À l'aide du **fork** du **Caffe de Intel** activé avec de nouvelles primitives Intel MKL sur un seul socket, le processeur Intel Xeon Phi 7200 offre une multiplication supplémentaire de l'accélération par 2 (**Figure 3**).



3 Performance d'apprentissage à un seul nœud Caffe/AlexNet

Une multiplication matricielle plus rapide

La multiplication matricielle matrice-matrice (GEMM) est une opération fondamentale dans de nombreuses applications scientifiques, d'ingénierie et d'apprentissage automatique. Il existe une demande constante pour optimiser cette opération.

Intel MKL offre des implémentations de GEMM parallèles et à haute performance. Pour fournir des performances optimales, l'implémentation de GEMM par Intel MKL transforme généralement les matrices d'entrée d'origine en un format de données interne mieux adapté à la plate-forme ciblée. Cette transformation de données (également appelé « packing » ou empaquetage) peut être coûteuse, en particulier pour les matrices d'entrée avec une ou plusieurs petites dimensions. Intel MKL 2017 introduit des interfaces de programme d'application de « packing » [S,D]GEMM pour éviter un empaquetage redondant. Ceci peut améliorer les performances sur certaines tailles de matrice et dans certains cas d'utilisation d'apprentissage automatique.

Les APIs permettent aux utilisateurs de transformer explicitement les matrices en un format interne empaqueté et de passer la matrice (ou les matrices) empaquetées à plusieurs appels GEMM. Avec cette approche, les coûts d'empaquetage peuvent être amortis sur plusieurs appels GEMM si les matrices d'entrée (A ou B) sont réutilisées entre ces appels, comme c'est le cas pour les réseaux récurrents de neurones.

Exemple

Trois appels GEMM présentés ci-dessous utilisent la même matrice A, alors que les matrices B / C différent pour chaque appel :

```
float *A, *B1, *B2, *B3, *C1, *C2, *C3, alpha, beta;
MKL_INT m, n, k, lda, ldb, ldc;
// initialise les pointeurs et les dimensions de la matrice (ignorés
pour plus de précision) sgemm("T", "N", &m, &n, &k, &alpha, A, &lda,
B1, &ldb, &beta, C1, &ldc);
sgemm("T", "N", &m, &n, &k, &alpha, A, &lda, B2, &ldb, &beta, C2, &ldc);
sgemm("T", "N", &m, &n, &k, &alpha, A, &lda, B3, &ldb, &beta, C3, &ldc);
```

Ici, la matrice A est transformée en format de données internes empaqueté au sein de chaque appel sgemmn. Le coût des 3 empaquetages de la matrice A peut être relativement élevé si n est petit (nombre de colonnes pour B / C). Ce coût peut être minimisé en empaquetant la matrice A une fois et en utilisant son équivalent empaqueté pour les trois appels GEMM consécutifs, comme démontré ci-dessous:

```
// alloue de la mémoire pour le format de données empaqueté
float *Ap;
Ap = sgemm_alloc("A", &m, &n, &k);
// transforme A en format empaqueté
sgemm_pack("A", "T", &m, &n, &k, &alpha, A, &lda, Ap);
// Les calculs SGEMM sont effectués à l'aide de la matrice A
empaquetée: Ap sgemm_compute("P", "N", &m, &n, &k, Ap, &lda, B1, &ldb,
&beta, C1, &ldc); sgemm_compute("P", "N", &m, &n, &k, Ap, &lda, B2,
&ldb, &beta, C2, &ldc); sgemm_compute("P", "N", &m, &n, &k, Ap, &lda,
B3, &ldb, &beta, C3, &ldc);
// libère la mémoire de Ap
sgemm_free(Ap);
```

Blog - Points essentiels

Linux* a 25 ans : Mener la révolution de l'Open Source

Par [IMAD SOUSOU >](#)

Où étiez-vous le 25 août 1991 ? Je ne me rappelle pas exactement où j'étais ou ce que je faisais, mais je peux vous assurer que cette journée a eu un impact profond sur ma vie et ma carrière. Il y a maintenant vingt-cinq ans, Linus Torvalds présentait le système d'exploitation Linux* et je pense que cette date du 25 août 1991 est l'une des plus importantes de l'histoire de la technologie.

Intel possède une relation fière et longue avec Linux. Linux a été lancé sur l'architecture Intel® (IA) en raison de son positionnement de meneur sur le marché, de ses performances exceptionnelles et d'une documentation consistante. Linux a grandi sur les plates-formes Intel® : du 386 aux puissants systèmes informatiques d'aujourd'hui. Notre implication s'étend au-delà du matériel ; Intel a été actif dans la communauté du logiciel Linux depuis le milieu des années 90. Notre première grande étape avec Linux a été le Intel® Dot.Station, lancé en 1999, ce fut non seulement le premier appareil Intel basé sur Linux, mais aussi notre premier système d'exploitation grand public.

Pour en savoir plus



< L'univers parallèle

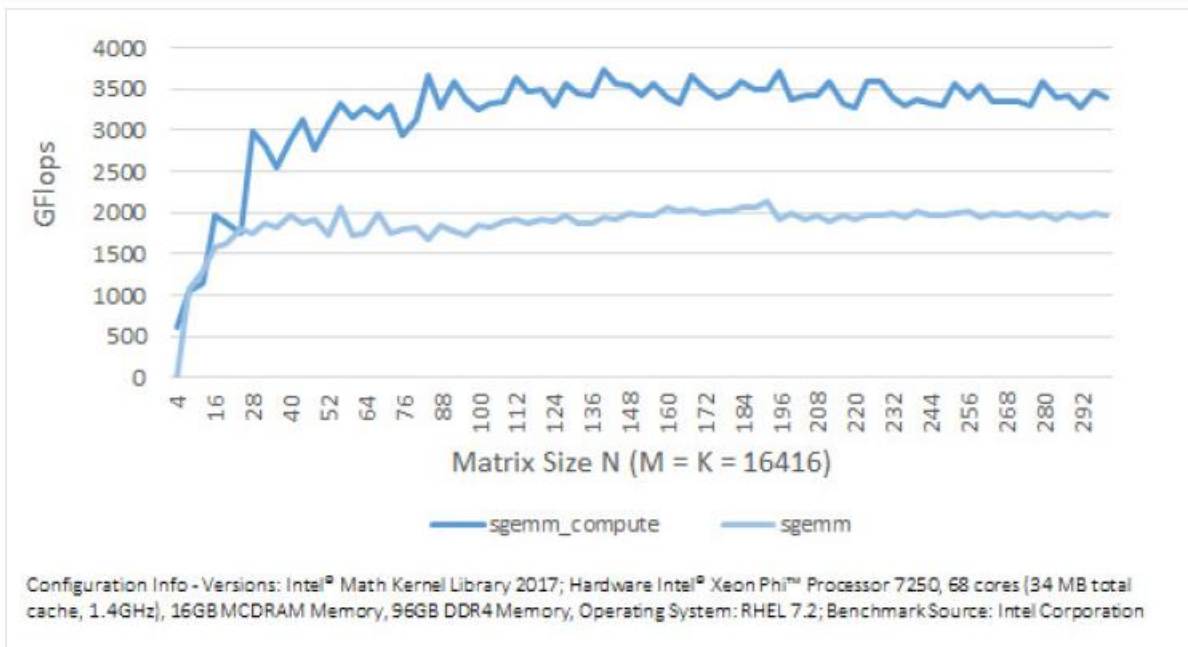
L'exemple de code ci-dessus utilise quatre nouvelles fonctionnalités introduites pour prendre en charge les API empaquetées pour GEMM : `sgemm_alloc`, `sgemm_pack`, `sgemm_compute` et `sgemm_free`. Tout d'abord, la mémoire requise pour le format empaqueté est attribuée à l'aide de `sgemm_alloc`, qui accepte un argument en caractère identifiant la matrice empaqueté (A dans cet exemple) et trois arguments entiers pour les dimensions de la matrice.

Ensuite, `sgemm_pack` transforme la matrice A originale dans format empaqueté Ap et effectue une mise à l'échelle alpha. La matrice A originale reste inchangée. Les trois appels `sgemm` sont remplacés par trois appels `sgemm_compute` qui fonctionnent avec des matrices empaquetées et supposent que $\alpha = 1.0$. Les deux premiers arguments en caractère de `sgemm_compute` indiquent que la matrice A est dans format empaqueté ("P"), et que la matrice B est dans un format majeur de colonne non transposé ("N"). Enfin, la mémoire allouée pour Ap est diffusée en appelant `sgemm_free`.

Les API d'empaquetage de GEMM éliminent le coût de deux empaquetages de la matrice A pour les trois opérations de multiplication matricielle présentées dans cet exemple. Ces API empaquetées peuvent être utilisées pour éliminer les coûts de transformation des données pour les matrices d'entrée A et/ou B si A et/ou B sont réutilisés entre les appels GEMM.

Performance

La figure 4 montre les gains de performances avec l'API d'empaquetage sur processeur Intel® Xeon Phi™ 7250. On suppose que le coût d'empaquetage peut être complètement amorti par un grand nombre d'appels SGEMM qui utilisent la même matrice A. La performance d'appels réguliers SGEMM est également fournie à titre de comparaison.



Notes d'implémentation

Il est recommandé d'appeler `gemm_pack` et `gemm_compute` avec le même nombre de threads pour obtenir de meilleures performances. Notez que s'il n'y a qu'un petit nombre d'appels GEMM qui partagent la même matrice A ou B, alors les API d'empaquetage n'amélioreront pas énormément les performances.

La routine `gemm_alloc` alloue une mémoire approximativement aussi grande que la matrice d'entrée d'origine. Ceci signifie que la mémoire requise par l'application peut augmenter considérablement pour une grande matrice d'entrée.

Les APIs GEMM d'empaquetage ne sont implémentées que pour SGEMM et DGEMM dans l'Intel MKL 2017. Elles sont fonctionnelles pour toutes les architectures Intel, mais ne sont optimisées que pour les processeurs Intel® AVX2 64 bits et supérieurs.

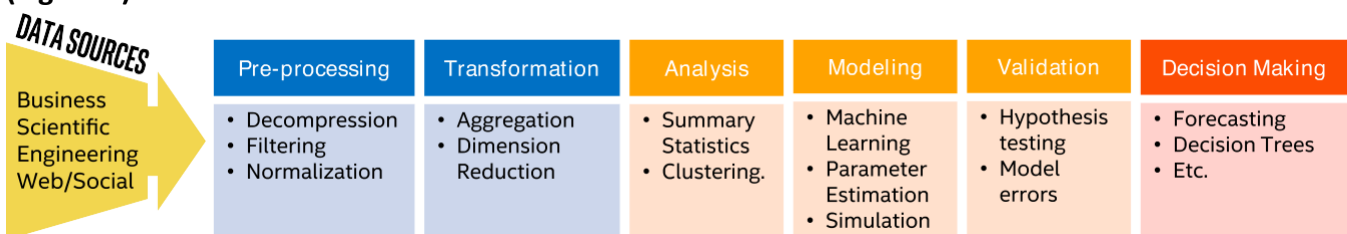
Intel DAAL

En grande partie, l'utilisation accrue et l'adoption d'algorithmes d'apprentissage automatique et d'apprentissage profond sont devenus possibles en raison de la croissance rapide des données disponibles pour l'apprentissage par algorithme. Le terme *big data* possède un certain nombre de définitions, nous traiterons le big data en termes de ses trois attributs principaux : variété, vitesse, et volume. Chaque attribut du big data nécessite des solutions de calcul spéciales, et Intel DAAL a été conçu pour répondre à toutes ces exigences.

Variété

Par *variété*, nous faisons référence aux nombreuses sources et types de données structurées et non-structurées qui créent des défis quant à l'extraction des données et leur analyse. Pour aborder cet attribut des big data, Intel DAAL introduit un composant de gestion de données en tant que partie essentielle de la bibliothèque. Ceci inclut des classes et des utilitaires pour l'acquisition de données, du prétraitement initial et de la normalisation, de la conversion de données en format numériques à partir d'une des nombreuses sources de données prises en charge et de la représentation de modèle. Étant donné que, en pratique, dans les applications analytiques, vous pouvez découvrir des goulots d'étranglement sur n'importe quelle partie de l'application, Intel DAAL se concentre sur l'optimisation de l'ensemble du flux de données. Il fournit des blocs de construction optimisés couvrant toutes les étapes de l'analyse des données : acquisition de données à partir d'une source de données, pré-traitement, transformation, exploration de données, modélisation, validation et prise de décision

(Figure 5).



De plus, Intel DAAL prend en charge les données homogènes et hétérogènes et effectue une conversion de données intermédiaire si nécessaire. Il prend également en charge les types de données denses et dispersés et fournit des algorithmes qui peuvent fonctionner avec des données bruitées.

Vélocité

La vitesse de calcul est essentielle dans l'environnement commercial actuel. L'un des aspects les plus importants du big data est qu'il se produit en temps réel et nécessite un temps de réponse rapide. Intel DAAL est conçu pour aider les développeurs de logiciels à réduire le temps de développement de leur application et de les livrer avec des performances améliorées. Intel DAAL aide les applications à faire de meilleures prévisions plus rapidement, en ajustant les ressources de calcul disponibles. Il s'appuie sur des éléments constitutifs Intel MKL et offre des optimisations algorithmiques supplémentaires pour générer des gains de performance maximum sur architecture Intel. Intel DAAL fournit des APIs pour les langages C ++, Java* et Python*. Ceci permet aux utilisateurs de prototyper rapidement leurs modèles et de les déployer facilement dans un vaste environnement de cluster.

Volume

Le big data implique d'énormes volumes de données, parfois si grand qu'il ne peut tenir en mémoire. Il est également très fréquent que l'ensemble des données soit réparti entre différents nœuds sur un cluster ou sur différents périphériques. Pour résoudre ceci, les algorithmes DAAL d'Intel prennent en charge les modes de calcul de traitement par lots, en ligne et distribués.

Dans le mode de traitement par lots, l'algorithme fonctionne avec l'ensemble des données pour produire le résultat final. Un scénario plus complexe se produit lorsque l'ensemble des données n'est pas disponible au moment même ou lorsque les données ne peuvent pas toutes être contenues dans la mémoire du périphérique, nécessitant ainsi les deux autres modes de traitement.

Dans le mode de traitement en ligne, l'algorithme traite un ensemble de données dans des blocs diffusés dans la mémoire de l'appareil, en mettant à jour progressivement des résultats partiels, finalisés lors du traitement du dernier bloc de données.

Dans le mode de traitement distribué, l'algorithme fonctionne sur un ensemble de données réparties sur plusieurs périphériques (nœuds de calcul). L'algorithme produit des résultats partiels sur chaque nœud, qui sont ensuite fusionnés dans un résultat final dans le nœud maître.

Étant donné qu'il existe de nombreuses plates-formes disponibles pour le calcul distribué, les algorithmes distribués d'Intel DAAL sont extraits de la technologie de communication inter-périphérique sous-jacente. Ceci permet d'utiliser la bibliothèque dans une variété de scénarios de calcul et de transfert de données multi-périphériques, y compris (sans s'y limiter) les environnements de cluster [MPI](#)-, [Hadoop](#)*, ou [Spark](#)*. Bien que l'utilisateur soit tenu d'implémenter une communication entre périphériques, Intel DAAL fournit des exemples qui montrent comment utiliser cette bibliothèque avec les plates-formes analytiques les plus courantes.

Utilisation d'Intel DAAL pour différents modes de calcul

Mode de traitement par lots

Tous les algorithmes DAAL d'Intel prennent en charge le mode de calcul de traitement par lot et il est relativement facile à utiliser. Dans le mode traitement par lots, seule la méthode de calcul d'une classe d'algorithme particulière est utilisée. Il suffit de sélectionner un algorithme à utiliser, de définir les paramètres d'entrée et de l'algorithme, d'exécuter la méthode de calcul et

```
// Crée un algorithme pour calculer la décomposition de Cholesky en
// utilisant la méthode par défaut
cholesky::Batch<> cholesky_alg;

// Définir les données d'entrée
cholesky_alg.input.set(cholesky::data, dataSource.getNumericTable())
// Exécuter le calcul
cholesky_alg.compute
();

// Accéder aux résultats du calcul
services::SharedPtr<cholesky::Result> res = cholesky_alg.getResult()
```

Mode de traitement en ligne

Certains algorithmes Intel DAAL permettent le traitement de données en blocs. L'API pour le mode de calcul de traitement en ligne est similaire au mode de traitement par lots, la différence vient du fait que vous devez exécuter la méthode de calcul à chaque fois que les données sont disponibles, puis vous devez enfin appeler `finalizeCompute` pour combiner les résultats. L'exemple ci-dessous explique comment utiliser Intel DAAL en mode de traitement en ligne avec un algorithme de décomposition en valeurs singulières (SVD) :



```
// initialisation de l'algorithm  
svd::Online<double, defaultDense> algorithm;  
  
// Traitement des blocs de données  
dans une boucle Status loadStatus;  
while((loadStatus = dataSource.loadDataBlock(nRowsInBlock)) == success)  
{  
    // définir les données d'entrée  
    algorithm.input.set( svd::data, dataSource.getNumericTable() );  
// Exécuter le calcul  
    algorithm.compute();  
}  
  
// Finaliser les calculs et récupérer les résultats du SVD  
algorithm.finalizeCompute();  
SharedPtr<svd::Result> res = algorithm.getResult();  
  
// Accès aux résultats  
printNumericTable(res->get(svd::singularValues), "Singular values:");
```

Mode de traitement distribué

Certains algorithmes Intel DAAL permettent le traitement de jeu de données distribués sur plusieurs périphériques. En fonction de l'algorithm, vous pouvez avoir besoin d'appliquer un schéma différent pour le calcul. Ceci peut aller d'un simple MapReduce, où la première étape est exécutée sur les nœuds locaux et l'étape finale sur le nœud maître, à des schémas plus complexes comme un siteReduce-Map. Le guide de programmation Intel DAAL inclut des exemples qui montrent comment utiliser Intel DAAL avec Hadoop, Spark, ou MPI. L'exemple ci-dessous montre comment utiliser Intel DAAL en mode de traitement distribué avec un algorithm d'analyse en composantes principales (PCA) :

Étape 1 : Sur les nœuds locaux (Mapper)

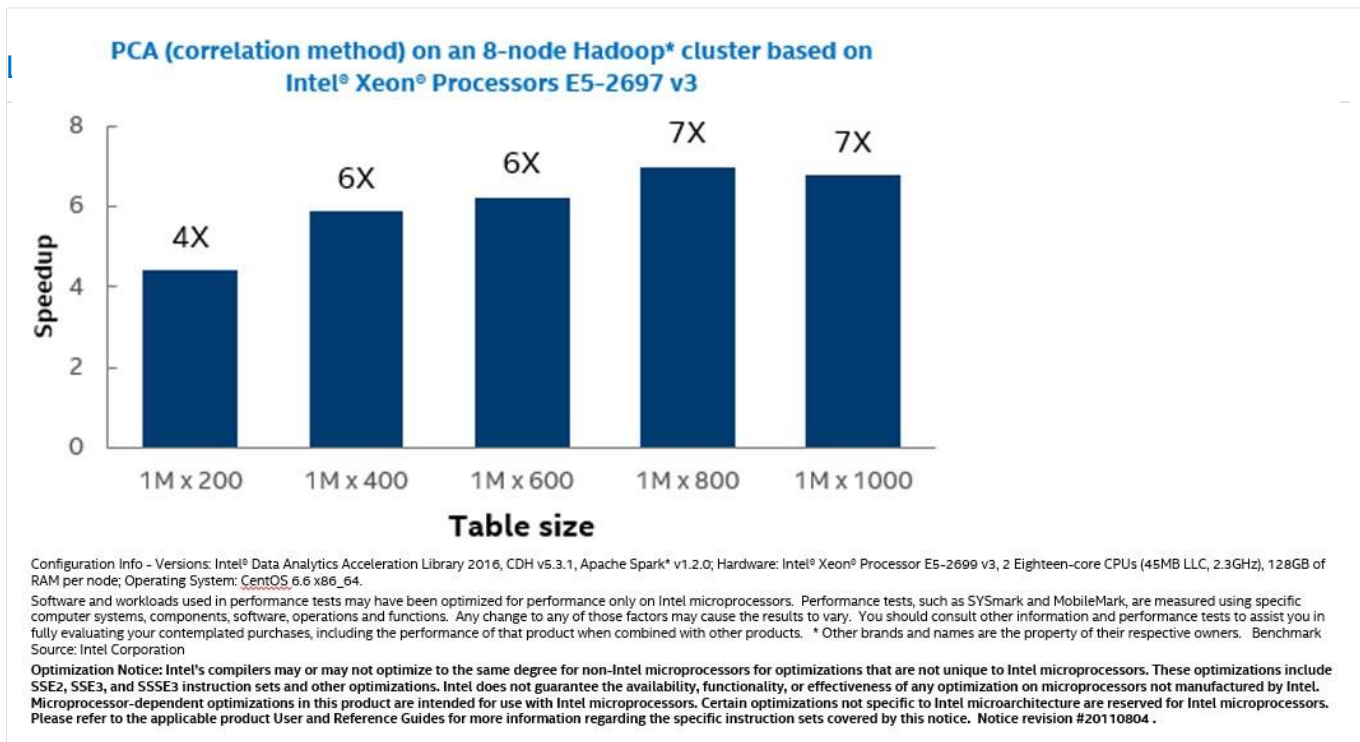
```
< // Création d'un algorithme permettant de calculer la décomposition PCA en
utilisant la méthode SVD sur des nœuds locaux
DistributedStep1Local pcaLocal = new
DistributedStep1Local(daalContext, Double.class, Method.svdDense);
// Définir les données d'entrée
pcaLocal.input.set( InputId.data, ntData );
// Calculer le PCA sur les nœuds locaux
PartialResult pres = pcaLocal.compute();
// Préparation des données pour envoi au nœud maître
context.write(new IntWritable(0), new WritableData(index, pres));
```

Étape 2 : Sur le nœud maître

```
// Création d'un algorithme permettant de calculer la décomposition PCA en
utilisant la méthode SVD sur un nœud maître
DistributedStep2Master pcaMaster = new DistributedStep2Master(daalContext,
Double.class, Method.svdDense);
// Ajouter des données d'entrée des nœuds
locaux au fil de l'eau for (WritableData value
: values) {
    PartialResult pr = (PartialResult)value.getObject(daalContext);
    pcaMaster.input.add( MasterInputId.partialResults, pr);
}
// Calcule du PCA sur le
nœud maître
pcaMaster.compute();
// Finaliser les calculs et récupérer les résultats
PCA Result res = pcaMaster.finalizeCompute();
```

Données de performance

La figure 6 affiche les gains de performance de la bibliothèque Intel Data Analytics Acceleration (Intel DAAL) par rapport à Spark MLlib sur un cluster Hadoop à 8 nœuds.



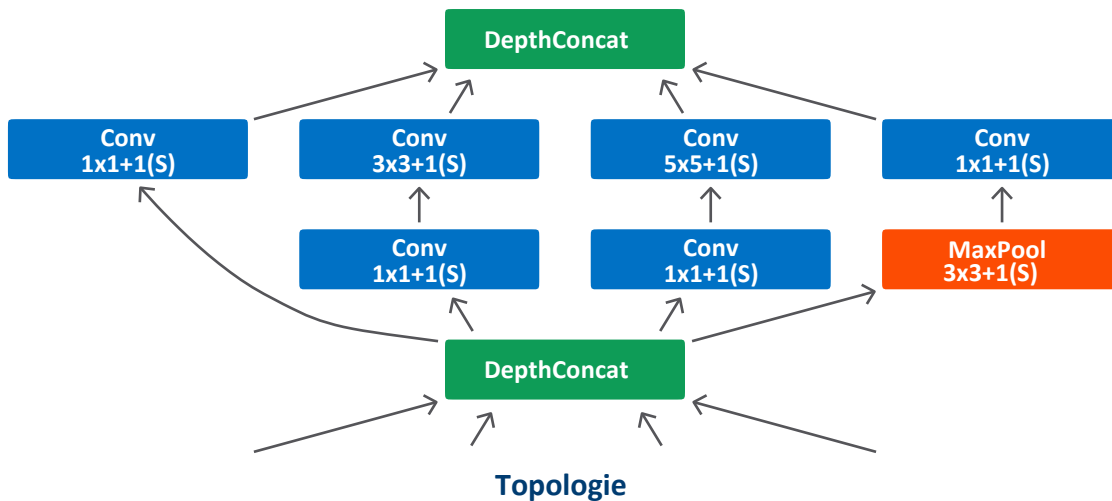
6 Performance PCA en utilisant Intel® DAAL versus Spark * MLlib sur les architectures Intel®

Quoi de neuf dans Intel DAAL 2017

Intel DAAL 2017 introduit un certain nombre de nouvelles fonctionnalités, notamment la mise en œuvre de la fonctionnalité du réseau de neurones. Ceci consiste en un certain nombre de composants qui permettent à l'utilisateur de construire et d'exécuter facilement diverses topologies de réseaux neuronaux allant de la classification d'image et du suivi d'objet aux prévisions de marché financier. Pour prendre en charge ces différents cas d'utilisation, Intel DAAL présente plusieurs composants essentiels des réseaux de neurones (**Figure 7**) :

- **Couche** : Éléments constitutifs d'un NN (réseau de neurones), versions avant et arrière d'une seule couche.
- **Topologie** : La structure pour représenter la configuration NN (p. ex. AlexNet).
- **Modèle** : Un ensemble de couches connectées en fonction de la topologie définie, leurs paramètres (pondérations et contraintes) et les routines de service.
- **Solutionneur d'optimisation** : Mises à jour de la pondération et des contraintes après passe-avant passe-arrière selon la fonction d'objectif spécifiée.
- **NN (Réseau de neurones)** : Pilote qui gère le flux de calcul du NN. Durant l'apprentissage, le pilote exécute des passe-avants et passe-arrières sur la topologie et met à jour les paramètres en utilisant le solutionneur d'optimisation. Au cours de la notation, il renvoie les résultats de prédiction en appliquant le passage avant.
- **Structure de données multidimensionnelles (tenseur)** : Structure utilisée pour représenter des données complexes (p. ex. des flux d'images 3D).

Réseau de neurones



Couche 1

Couche 2

Couche 3

Modèle

Algorithme d'optimisation

7 Éléments constitutifs pour les réseaux de neurones Intel® DAAL

Avec les mêmes API que les algorithmes d'apprentissage automatique traditionnels, il devient relativement facile pour un scientifique de comparer l'exactitude d'un modèle créé avec des réseaux de neurones à celui d'algorithmes d'apprentissage automatique traditionnels (SVM) ou de régression linéaire, tout en obtenant d'excellentes performances sur architecture Intel. Les fonctionnalités de la bibliothèque révèlent une opportunité de combiner des réseaux de neurones avec d'autres algorithmes (par exemple, injecter le modèle de Neural Net dans des algorithmes de boosting en tant qu'apprenants faibles).

Relever les défis de l'apprentissage automatique et du Big Data

S'attaquer à l'analyse du big data et à l'apprentissage automatique exige généralement aux développeurs de traiter avec différents langages de programmation, différents outils et bibliothèques pour résoudre leurs problèmes. Il est assez courant que les données proviennent d'une grande variété de sources et soient hétérogènes, dispersées ou bruitées. Les langages de script tels que Python et d'autres sont couramment utilisés pendant la phase de prototypage, tandis que les langages C / C ++ sont appliqués pour l'optimisation des parties les plus critiques du flux de données. Les données réelles arrivent au fil du temps et/ou ne tiennent généralement pas dans la mémoire d'un ordinateur, l'utilisation de systèmes distribués comme MPI, Hadoop ou Spark est donc nécessaire pour un traitement efficace des données.

À cause de tout ceci, le processus de développement, de test et de support nécessite pour les développeurs d'avoir des connaissances dans divers domaines. Il faut du temps pour intégrer de nombreuses solutions sous une seule application. Par conséquent, le temps de développement augmente considérablement et l'adoption des analyses de big data est retardée.

Intel DAAL est conçu pour répondre à tous ces défis et couvre la plupart des cas d'utilisation possibles rencontrés par un développeur qui travaille avec des big data. La bibliothèque se concentre sur l'optimisation de l'ensemble du flux de données, pas uniquement la partie algorithmique, et fournit des éléments constitutifs optimisés qui couvrent toutes les étapes de l'analyse de données : acquisition à partir d'une source de données, prétraitement, transformation, exploration de données, modélisation, validation et prise de décision. Il est possible d'étendre un modèle à partir d'un seul nœud vers un plus grand cluster tout en utilisant les mêmes API. Intel DAAL, avec Intel MKL, permet de libérer le pouvoir du big data et de l'apprentissage automatique.



**En savoir plus sur la bibliothèque
d'accélération des données Intel® Data
Analytics(intel®dAAI)>**