.Net 10 : tout comprendre des optimisations JIT et dans le code

Par: francoistonic | jeu, 09/10/2025 - 09:06

.net (/actualites/net)

C'est un des posts les plus attendus de la communauté .Net : performance improvements de Stephen Toub de Microsoft. Avec la sortie de .Net 10, la performance revient naturellement un sujet pour les développeurs. Dans son très long post, Stephen revient sur les améliorations réalisées depuis .Net Core 2.0 jusqu'à maintenant. Ce sont des centaines d'améliorations qui ont été injectées dans .Net 9 et .Net 10. Pas de grandes annonces mais de petits ajustements qui font la différence. Stephen compare .Net 9 et 10.

Le benchmark s'appuie sur plusieurs éléments : un code commun et le fonctionnement en JIT, l'allocation des objets, la deabstraction (et la notion d'abstraction penalty). Sur un code simple, .Net 10 améliore sur tous les éléments : le temps de traitement, la taille du code, l'allocation mémoire. Le rôle du JIT est crucial pour les traitements et les performances en sortie. Mais c'est aussi la manière dont le code est structuré qui impact le pré-traitement et le traitement JIT.

Un des points sensibles de .net est la notion d'interfaces et l'abstraction qui va avec. Mais cela peut pénaliser le traitement JIT. Par exemple, sur un SumEnumerable, les gains sont d'environ 20 %, 70 % sur un SumForLoop.

Stephen prend des exemples très concrets : avec les codes et les explications et pourquoi .Net traite ainsi et comment le code et le JIT se comportent. A chaque problème de performance, il dit pourquoi.

Au-delà des chiffres purs, le post est passionnant car il explique comment des choix structurels du langage impactent directement les performances. Le Inlining est intéressant sur ce point. Il remplace le simple appel à une fonction avec une copie de l'implémentation de la fonction. Si le inlining paraît une solution "simple", son implémentation a permis d'optimiser les traitements du code, d'améliorer la sécurité. Dans ce cas précis, Stephen précise aussi que parfois il faut être faire attention à l'inline traité par le JIT. A vouloir faire de l'inline pour tout, il peut arriver qu'il n'y ait aucun bénéficie dans le traitement. C'est là qu'intervient la notion "heuristique".

Autre point abordé : le compilateur JIT génère de l'assembleur depuis un langage intermédiaire émis par le compilateur C#. Les couches de codes peuvent impacter les performances et le bon fonctionnement du code. Sur .Net, le PGO (Profile-guided Optimization) dynamique dans le JIT a un impact direct sur les performances de la compilation et donc le code final.

Stephen évoque tous les éléments liés à l'optimisation et aux performances : les extensions matérielles, les architectures CPU (via les instruction sets), le native AOT, le garbage collector, la machine virtuelle .Net (qui ne bouge pas fondamentalement en .Net 9 et 10), les threads (20 % d'optimisation). Il revient aussi sur la réflexion, les énumérations, le parsing réseau, le regex, JSON (un nette amélioration sur un Set).

Si vous voulez comprendre en profondeur .Net 10 et comment la plateforme est optimisée, prenez le temps de bien lire le post : https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-10/ (https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-10/)

Préparez votre dose de café et au moins 2 à 3h.