Resumo

Para se obter os benefícios das mais atuais evoluções em *hardware* passa a ser necessário escrever softwares concorrentes e paralelos. Diversas linguagens, clássicas e atuais, implementam um ou mais tipos de modelos de paralelismo. O presente trabalho estudou o comportamento de quatro dessas linguagens – Go, Julia, Python e C/OpenMP – em um contexto perfeitamente paralelo de uso intenso de CPU, utilizando-se como ferramenta de comparação o algoritmo de obtenção do conjunto de Mandelbrot. Foi possível mostrar que, para as linguagens estudadas, pequenas adições ao código podem trazer um grande ganho em desempenho. Para a situação específica envolvendo C/OpenMP, executando paralelamente em 8 núcleos com escalonamento dinâmico de tarefas e com matriz de entrada de tamanho 14000×10000, verificou-se um desempenho quase 7x maior para uma adição de apenas 2 linhas de código em relação à execução em modo sequencial. Os programas implementados em Julia, utilizando paradigma funcional, mostraram alto desempenho mesmo quando executados em modo sequencial, e tiveram um *speedup* em latência considerável com a adição de 2 linhas de código. O desempenho comparável a C, a simplicidade do código e a existência de diversas bibliotecas otimizadas fazem de Julia a melhor escolha geral dentre as linguagens estudadas. No caso específico em que há necessidade de se iniciar muitos processos paralelos, Go passa a ser uma boa opção devido à leveza de suas rotinas. As linguagens mais atuais e as extensões mais recentes para linguagens clássicas estão, a cada dia, tornando o paralelismo mais acessível mesmo aos programadores mais leigos. Espera-se mostrar, com este trabalho, que todos os programadores estão convidados a experimentar o paralelismo em sua linguagem de preferência e, possivelmente, obter proveito dele em sua rotina diária.

Palavras-chave: paralelismo; desempenho; tamanho do código; uso intenso de CPU.