1 Introdução *(Motivação)*

Das tecnologias no campo dos computadores, a linguagem *Java* foi das que teve maior efeito no mercado. Desde páginas *web* usando *applets Java*, até aplicações desenvolvidas completamente nesta linguagem, a linguagem *Java* tem vindo a honrar o seu mote de "*Write Once, Run Anywhere*". Existem opiniões de que a linguagem *Java* é apenas mais uma das muitas linguagens de programação, no entanto, a *Sun Microsystems* trouxe[1] a linguagem *Java* à engenharia de hardware, com a introdução de uma série de *chips* que são capazes de executar aplicações *Java* com uma muito melhor relação preço qualidade que a maioria dos microprocessadores das arquitecturas actuais. Como é que a *Sun* consegue isto ? Executando *bytecode java* nativamente.

2 A plataforma *Java*, a máquina virtual *Java* e *Bytecode Java*

Existem várias tecnologias associadas a *Java*, que no seu conjunto é chamado de "*Plataforma Java*". Nestas, incluí-se a *Linguagem Java* que é uma linguagem de programação orientada a objectos (semelhante a C++), mas com algumas *nuances* que a distinguem, como *thread sincronization*, *automatic garbage collection* e um potente conjunto de APIs. A *Java Virtual Machine* representa o núcleo da plataforma *Java*. Tal, como numa linguagem normal, a aplicação tem de ser compilada, no entanto, em *Java*, esta é compilada em *bytecode*, e não para uma arquitectura específica. Assim, a *Java Virtual Machine* executa então *bytecodes*. É isto que torna interessante a microarquitectura *picoJava* que se explica na secção seguinte. A *Java Virtual Machine* além de definir uma linguagem de programação, basicamente define também uma arquitectura de computadores, dado que define como variáveis e métodos são emparelhados na memória, que *opcodes* e variáveis são permitidos, o tamanho das instruções, etc. Basicamente, descreve como uma máquina deve ser, de forma a correr programas em *Java*. Resumidamente, define um interpretador de *Java* em software, mas também define como é que um processador deve actuar.

[1] No último trimestre de 1997

3 A microarquitectura *picoJava*

A microarquitectura *picoJava*, desenvolvida também pela *Sun Microsystems*, é voltada para a execução eficiente de *Java* directamente do *hardware*, tendo como principal mercado, os sistemas embutidos. Esta microarquitectura possibilita a execução, com alto desempenho, de aplicativos *Java*, em processadores relativamente baratos, para os quais (e para o mesmo nível de *performance*) seriam necessários processadores de alto desempenho com compiladores JIT[2]. Fica assim aberta uma vasta gama de aplicações embutidas para o uso de *Java*. Por possuir diversos mecanismos de verificação de segurança e acesso à memória, a linguagem *Java* facilita a programação de sistemas embutidos. A alta portabilidade é outra característica sedutora desta linguagem. Todas as aplicações *Java* são compiladas em *bytecodes*, que constituem, por sua vez, o conjunto mínimo de instruções a serem implementadas por um processador *Java*.
O desenvolvimento da microarquitectura *picoJava* levou em conta principalmente os aspectos de suporte à linguagem *Java*, na forma de *bytecodes*, e a eficiência da execução dos seus programas. Os processadores resultantes podem, inclusivé, implementar em *hardware* funções do sistema operativo e da *Java Virual Machine*, como *automatic garbage collection*, e *thread synchronization*, o que aumenta a eficiência de execução. Além de *Java*, os microprocessadores *picoJava* podem ainda executar qualquer linguagem que seja compilada para *bytecodes*[3].
Outra vantagem de *picoJava* sobre processadores *RISC* convencionais é o tamanho relativamente pequeno[4] do código resultante. Desta forma, as exigências de memória[5] para seu armazenamento são menores, o que favorece novamente a concepção de sistemas embutidos.
Algumas das características da *picoJava* são claramente determinadas para atender às necessidades destes sistemas, como economia de energia (embora não conseguida nas versões iniciais dos

[2] Just In Time

[3] A microarquitectura *picoJava-II* possibilita também a execução de código C/C++

[4] O tamanho médio das instruções é de 1.8 *bytes*

[5] A inclusão do compilador *JIT* na aplicação baseada na arquitectura *RISC* aumenta as exigências de memória por um factor médio de 3

microprocessadores desta microarquitectura, que apresentavam uma potência dissipada entre 3 e 4 *Watts*), simplicidade e a possibilidade de excluir certos componentes de *hardware* não críticos, como, por exemplo, a unidade de vírgula flutuante. Outras características, não tão óbvias, também são fortemente influenciadas pelos requisitos de previsibilidade dos sistemas de tempo real: o *pipeline* de instruções relativamente curto, favorece a determinação de parâmetros temporais, tão importantes para o desenvolvimento em tempo real. As caches também foram desenhadas com o intuito de, além de adequar-se às características da *Java Virtual Machine*, permitir definir mais facilmente o desempenho do sistema, ou seja, aumentar a previsibilidade deste.

Na verdade, a microarquitectura *picoJava* não é como uma arquitectura de microprocessador que normalmente se encontra no mercado. É flexível e pode ser modificada de acordo com as necessidades de quem a desenvolve. Desta forma, custos no desenvolvimento de um microprocessador específico para os requisitos de um determinado sistema podem ser cortados drasticamente.

A especificação em *Verilog*, os simuladores e a documentação permitem um estudo aprofundado da microarquitectura. A microarquitectura pode, inclusivé, ser utilizada para testar e simular modificações na arquitectura, o que é interessante do ponto de vista do estudo de arquitecturas de pilha e, em especial, da *picoJava*.

A microarquitectura *picoJava* tenta diminuir os problemas encontrados em arquitecturas voltadas para linguagens de programação e orientadas à pilha[6], através de suporte por *hardware* para tais mecanismos.

As características da linguagem *Java* e a simplicidade da microarquitectura *picoJava* permitem vislumbrar diversas aplicações e projectos. Por exemplo, um multiprocessador num só *chip* poderia executar múltiplas máquinas virtuais *Java* em paralelo e seria usada, por exemplo, em *Network Computers* executando diversas *applets*. Uma arquitectura *multi-thread* desenvolvida a partir de *picoJava* permitiria aproveitar as múltiplas *threads* normalmente encontradas em aplicações *Java*.

A microarquitectura *picoJava* pode também ser modificada, retirando algumas características do tempo real, para torná-la uma arquitectura voltada para processamento de alto desempenho com exploração de paralelismo ao nível de instrução (ILP). Porém, as alterações seriam tão grandes que, na verdade, provavelmente surgiria uma nova microarquitectura.

4 Conclusão

Não é completamente inédita a ideia de processadores capazes de executar uma linguagem de alto nível nativamente[7], no entanto, todas as tentativas em arquitecturas deste tipo, não tiveram implementações a nível de mercado. A linguagem *Java* é, no entanto, diferente da maioria das linguagens de programação, pelo que esta implementação da *Sun Microsystems* tem o seu mérito. Algo, no entanto, tem faltado para que esta microarquitectura produza resultados no âmbito de aplicações embutidas para as quais ela se destina primariamente. A *Sun* vendeu algumas licenças para a utilização desta microarquitectura, no entanto, as aplicações reais e em mercado são escassas. Como concorrente encontra a arquitectura *StrongArm*, em que a *Advanced Risc Machines* desenvolveu novas especificações para aplicações embutidas e linguagens baseadas em *stack*, como *Java* e *PostScript*, que embora não permitam uma execução mais rápida de *bytecode Java*, garantem um melhor gerenciamento dos recursos do sistema e um uso mais eficiente da cache. Outros processadores foram aperfeiçoados tendo em conta *Java*, como a arquitectura *Rx000* da *Mips (Silicon Graphics)*. Assim, o mercado da microarquitectura *picoJava* encontrou-se parcialmente ameaçado. Por outro lado, actualmente a indústria dá mais valor a aplicações nativas, menosprezando a independência de plataforma e mecanismos de segurança fornecidos pela *picoJava*, pelo que, o nicho de mercado de aplicações embutidas e de baixo consumo ainda escoa para arquitecturas diferentes desta que, muito resumidamente, aqui se analisou.

[6] Cerca de $\frac{1}{3}$ das instruções em *Java* são relativas a *push* e *pop* da pilha

[7] Nomeadamente, e em nível académico, para *LISP* e *Smalltalk*, tendo como conclusões principais que implementações de software executadas em *chips RISC* ofereciam uma performance francamente superior