Perceived Issues with Interaction Nets

written by Eashan Hatti on Functor Network
original link: https://functor.network/user/721/entry/466

I really like the idea of interaction nets, which means I'm obligated to learn at least a little about them. It seems to me that they have fundamental issues that make them impractical, so I'll post them here and hope someone can prove me wrong ;)

Interaction nets are seen as promising for huge performance gains because of two things:

1. They allow for optimal evaluation
2. They allow for massive parallelism

Let's work through the two and find the issues.

Optimal Evaluation is not Necessarily Optimal Performance

Interaction nets are said to allow for optimal evaluation. The first thing we have to do is actually define this, because it does not mean "optimal performance". Optimal evaluation refers to the very specific concept of work – work is time spent performing a computation. If evaluation is optimal, multiple computations that produce the same result will not be evaluated multiple times – the results will be shared between all of the computations so that they don't have to repeat a task. The problem is that you gain this optimal time complexity in exchange for horrible space complexity, the brackets and croissants used to implement the "bookkeeping" that higher-order sharing requires can build up and result in exponential memory consumption.

Alright, so interaction nets inherently require enormous space blowups, but what if we just say that's fine? We have huge amounts of memory at our disposal and we can probably employ some dumb optimizations to make the space consumption at least manageable. The problem here becomes that the optimal evaluation that interaction nets give us just isn't that useful in the first place.

Let's add some nuance to the first paragraph, when do we actually need optimal evaluation? We need it whenever we have lambdas – any sort of higher-order terms. The problem that interaction nets really solve is sharing under lambdas. Languages such as Haskell already do a lot of sharing! Computations are memoised and so only performed once, except for one main case: Namely, when they are inside a function – function bodies have to be duplicated. Interaction nets do not have this issue, and so achieve optimal sharing.

The problem with this is a very concrete one – in practice, the performance gains of interaction nets just don't match up to the traditional optimizations we have for functions. The place interaction nets start to really show massive performance gains is in massively higher order programs, the kinds of programs you only get if you use stuff like Church encodings, and these higher order programs appear very rarely in practice. Because of this, the theoretical performance gains of interaction nets never have a chance to actually be realized.

In practice, you don't want to optimally evaluate stuff like basic addition. You can get much higher performance by just using native integers and machine instructions. And the more and more you do this, the less and less you exploit the actual benefits of interaction nets.

Applicability of Massive Parallelism

The second benefit of interaction nets that people tout is massive parallelism. There are some easy criticisms of this that I'll start with, namely that CPUs just can't exploit this property of interaction nets because they don't have enough cores. So of course, you move to GPUs and FGPAs, but by doing that you're limiting the applicability of interaction nets – you can't use them for as many tasks because they have to be used on specialized hardware.

But an even more fundamental thing here is that we can already realize this massive parallelism in more traditional languages. The lambda calculus already can be massively parallelized to somewhat the same effect as interaction nets, see Futhark. The advantage of interaction nets is that fundamentally *every* computation can be parallelized, but we run into the same issue as with optimal evaluation – that the kinds of workloads that that property benefits just don't appear in practice, and the bookkeeping runtime costs of doing all that parallelism offset the potential gains.

TL;DR: Interaction nets allow us to parallelize and compute optimally on the most fine-grained levels possible, but the space/hardware overhead of doing that means that all but the most utterly massive computations end up being slower in practice as opposed to traditional methods.