We would like to thank the reviewers for useful comments and suggestions for improvement – we shall use them to improve the quality of the paper. Meanwhile we would like to address some questions and misunderstandings.

The library source code, the discussed Pivot application, and the timing data on different combinations of architectures, compilers, and optimizations are available at <https://parasol.tamu.edu/~yuriys/pm/>. The source code represents a fully functional subset of a greater pattern-matching library relevant to this paper. We avoid discussing any pattern matching facilities though in order to remain focused on the implementation of type switch. Similarly, we do not accentuate the fairly straightforward library implementation with macros and template meta-programming, because the technique is well suited for compiler implementation as well. For the clarity of presentation, however, we shall indicate visually in the examples of section 3 what parts of the generated code are coming from what macros. The example on page 1 is complete and requires no other definitions from the user besides the inclusion of a header. A similar example is available with the library’s code above in file example02.cpp.

In the comparison of performance to visitors (4.1) and to similar facilities of Haskell and OCaml (4.3), the 101 experiments time 1000000 iterations each. The first serves as a warm-up, during which optimal caching parameters are inferred. In the remaining 100 experiments, the measured number of cycles per iteration is sorted and the median is chosen. We prefer median to average to account for possible OS interruptions by other applications – in our experiments 70-80 out of those 100 timings render exactly the same number, which is especially repeatable in diagnostic boot of OS, where the minimum of drivers and applications are loaded. Sorted timings can be found in Area\*.csv files produced by running any of the timing tests in the above source code.

The languages that use v-tables to implement run-time dispatch can use our technique to implement open type switch efficiently. Absence of multiple inheritance in a language can further improve performance and memory footprint by not requiring maintaining offsets and performing this-pointer adjustments. There is a significant amount of work on the use of dynamic compilation to improve general run-time dispatch and many of these techniques are applicable to type switching. Unlike general run-time dispatch, however, type switching has a fixed number of targets, which is what lets us lay out the code efficiently even at compile time. The dynamic compilation can be used to achieve optimal or even perfect hashing.

All three reviewers ask for more details about the implementation of the matching library. The library is fairly straightforward C++ reflecting the algorithms described and are publicly available in source form (https://parasol.tamu.edu/~yuriys/pm/). We will add a some explanation and a short example of generated C++.